

**AGENTS FOR EFFICIENT AND OPTIMAL SEARCH  
ON SEMANTIC WEB**



By

NC Sara Rehmat

NC Urooj Saeed

NC Amina Khan

NC Fatima Naeem

Submitted to the Faculty of Computer Science

National University of Sciences and Technology, Rawalpindi in partial fulfillment for the  
requirements of a B.E Degree in Computer Software Engineering

MARCH 2008

**ABSTRACT**

**AGENTS FOR EFFICIENT AND OPTIMAL SEARCH**

**ON SEMANTIC WEB**

All advances in the field of science and technology aim at one main purpose of providing humans as much convenience as possible. To contribute towards this aim, many automation techniques have been developed in various fields to achieve quality performance with minimum human intervention.

World Wide Web developed in 1980s is the universe of network-accessible information. But most of the Web's content today is designed for humans to read, not for computer programs to manipulate meaningfully. Computers can parse Web pages for layout and routine processing but in general, computers have no reliable way to process the semantics. The World Wide Web can be automated by putting semantics in it that can be understood by the computer programs thus enabling them to act on users' behalf resulting into a new form of web called Semantic Web.

Mobile agents are programs that can autonomously travel across a network and perform tasks on machines that provide agent hosting capability. The use of Mobile Agent reduces network load as instead of downloading large amount of data for processing, the processing is transferred to where data lies.

In the proposed project, the Mobile Agent searches for a job on user's behalf on Semantic Web. The user provides it with his job preferences, on the basis of which the Mobile Agent searches on the Semantic Web, returning very precise and refined results to the user instead of the plethora of information which is returned in case of conventional search engines.

## **DEDICATION**

In the name of Allah, the Most Merciful, the Most Beneficent

To our parents, without whose unflinching support and unstinting cooperation, a work of  
this magnitude would not have been possible

## **ACKNOWLEDGMENTS**

We are eternally grateful to Almighty Allah for bestowing us with the strength and resolve to undertake and complete the project.

We gratefully recognize the continuous supervision and motivation provided to us by our Project Supervisor, Mr. Athar Mohsin. We are highly gratified to our co-supervisor Mr. Umar Mahmud for his continuous and valuable suggestions, guidance, and commitment towards provision of undue support throughout our thesis work. We are also grateful to our external supervisor Mr. Tauseef Rana for his constant guidance throughout the project. We are highly thankful to all of our professors whom had been guiding and supporting us through out our course and research work. Their knowledge, guidance and training enabled us to carry out this research work.

We would like to offer our admiration to all our classmates, and our seniors who had been supporting, helping and encouraging us throughout our thesis project. We are also indebted to the MCS system administration for their help and support.

We deeply treasure the unparallel support and tolerance that we received from our friends for their useful suggestions that helped us in completion of this project.

We are also deeply obliged to our families for their never ending patience and support for our mental peace and to our parents for the strength that they gave us through their prayers.

## TABLE OF CONTENTS

<b>LIST OF TABLES .....</b>	<b>xi</b>
<b>LIST OF FIGURES .....</b>	<b>xii</b>
<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>1.1 Preface.....</b>	<b>1</b>
<b>1.2 Project Vision .....</b>	<b>2</b>
<b>1.3 Proposed Solution .....</b>	<b>2</b>
<b>1.4 Aim of the Project .....</b>	<b>3</b>
<b>1.5 Organization of Project Report .....</b>	<b>4</b>
<b>2. LITERATURE REVIEW .....</b>	<b>5</b>
<b>2.1 Introduction.....</b>	<b>5</b>
<b>2.2 Agents .....</b>	<b>5</b>
<b>2.3 Mobile Agent .....</b>	<b>5</b>
<b>2.4 Advantages of Using Mobile Agent .....</b>	<b>5</b>
<b>2.4.1 Reduce the network load .....</b>	<b>6</b>
<b>2.4.2 Overcome network latency .....</b>	<b>6</b>
<b>2.4.3 Asynchronous and Autonomous execution.....</b>	<b>6</b>
<b>2.4.4 Dynamic Adaptation and Robustness .....</b>	<b>6</b>
<b>2.4.5 Heterogeneity.....</b>	<b>7</b>
<b>2.5 JADE-Tool for Developing Agents .....</b>	<b>7</b>
<b>2.5.1 Important Terms of JADE .....</b>	<b>7</b>
<b>2.5.2 Agent Communication.....</b>	<b>8</b>

<b>2.5.3 Reasons of using JADE.....</b>	<b>8</b>
<b>2.6 Semantic Web.....</b>	<b>8</b>
<b>2.7 Semantic Web Technologies.....</b>	<b>8</b>
<b>2.7.1 XML .....</b>	<b>9</b>
<b>2.7.2 RDF .....</b>	<b>9</b>
<b>2.7.3 Ontology.....</b>	<b>9</b>
<b>2.7.4 OWL.....</b>	<b>10</b>
<b>2.7.4.1 Reasons of using OWL .....</b>	<b>10</b>
<b>2.7.5 Protégé .....</b>	<b>10</b>
<b>2.7.5.1 Reasons of using Protégé .....</b>	<b>11</b>
<b>2.7.6 JENA .....</b>	<b>11</b>
<b>2.7.7 Pellet.....</b>	<b>11</b>
<b>2.7.7.1 Difference between Pellet and JENA Query Engines .....</b>	<b>12</b>
<b>2.7.7.2 Using Pellet in JAVA Applications .....</b>	<b>12</b>
<b>2.7.7.3 Using Pellet with JENA .....</b>	<b>12</b>
<b>2.8 Conclusion .....</b>	<b>13</b>
<b>3. SYSTEM ANALYSIS.....</b>	<b>14</b>
<b>3.1 Introduction.....</b>	<b>14</b>
<b>3.2 Project Scope .....</b>	<b>14</b>
<b>3.3 Requirements Specification.....</b>	<b>15</b>
<b>3.3.1 External Interface Requirements .....</b>	<b>15</b>
<b>3.3.1.1 User Interface .....</b>	<b>15</b>
<b>3.3.1.2 Software Interfaces .....</b>	<b>16</b>

<b>3.3.2 Major Functional Requirements .....</b>	<b>16</b>
<b>3.3.2.1 Acquiring User’s Preferences and Personal Data.....</b>	<b>16</b>
<b>3.3.2.2 Creating Semantic Websites .....</b>	<b>16</b>
<b>3.3.2.3 Registration of Websites with Directory .....</b>	<b>17</b>
<b>3.3.2.4 Directory Lookup by Mobile Agent .....</b>	<b>17</b>
<b>3.3.2.5 Cloning to the Relevant Websites.....</b>	<b>17</b>
<b>3.3.2.6 Processing of Semantic Websites by Mobile Agent Clone .....</b>	<b>17</b>
<b>3.3.2.7 Returning the Results .....</b>	<b>17</b>
<b>3.3.2.8 Dropping the user’s CV .....</b>	<b>18</b>
<b>3.3.3 Major Non-Functional Requirements.....</b>	<b>18</b>
<b>3.3.3.1 Accuracy .....</b>	<b>18</b>
<b>3.3.2.2 User friendly GUI .....</b>	<b>18</b>
<b>3.4 Usecase Diagram .....</b>	<b>19</b>
<b>3.5 Domain Model .....</b>	<b>19</b>
<b>3.6 Conclusion .....</b>	<b>20</b>
<b>4. SYSTEM DESIGN.....</b>	<b>22</b>
<b>4.1 Introduction.....</b>	<b>22</b>
<b>4.2 Architectural Diagram .....</b>	<b>22</b>
<b>4.3 High Level Diagram.....</b>	<b>23</b>
<b>4.4 Low Level Diagram.....</b>	<b>24</b>
<b>4.4.1 Agent Platform .....</b>	<b>24</b>
<b>4.4.2 Directory .....</b>	<b>25</b>
<b>4.4.3 User’s Machine.....</b>	<b>25</b>

4.4.4 Website Hosting Machine.....	25
4.4.5 Search by Mobile Agent .....	26
4.5 Class Diagram .....	27
4.6 Data Flow Diagram.....	28
4.7 Interaction Diagram .....	29
4.8 Conclusion .....	32
5. IMPLEMENTATION .....	33
5.1 Introduction.....	33
5.2 Implementation Language .....	33
5.3 Distribution of classes with respect to Modules .....	34
5.3.1 User’s Machine.....	34
5.3.1.1 Initiating Mobile Agent .....	34
5.3.1.2 Job Seeker GUI for acquiring his Job Criteria.....	35
5.3.1.3 Job Seeker GUI for acquiring his Personal Data.....	36
5.3.1.4 Manipulating Personal Data .....	37
5.3.1.5 Manipulating Job Criteria .....	37
5.3.1.6 User’s Personalized Mobile Agent.....	37
5.3.1.7 Query Engine.....	38
5.3.1.7.1 run(String queryString) .....	39
5.3.1.7.2 runQuery().....	39
5.3.2 Directory .....	39
5.3.3 Website Hosting Machine.....	40
5.3.3.1 Initiating Register Agent .....	40



<b>5.3.3.2 Job Employers GUI for Job Registration .....</b>	<b>40</b>
<b>5.3.3.3 Registration of Jobs .....</b>	<b>41</b>
<b>5.3.3.4 Ontology .....</b>	<b>42</b>
<b>5.3.3.4.1 First Level .....</b>	<b>42</b>
<b>5.3.3.4.2 Second Level .....</b>	<b>42</b>
<b>5.3.3.4.3 Third Level .....</b>	<b>43</b>
<b>5.3.3.4.4 company .....</b>	<b>43</b>
<b>5.3.3.4.5 location .....</b>	<b>43</b>
<b>5.3.3.4.6 pay .....</b>	<b>43</b>
<b>5.3.3.4.7 qualification .....</b>	<b>43</b>
<b>5.4 Conclusion .....</b>	<b>44</b>
<b>6. TESTING .....</b>	<b>45</b>
<b>6.1 Introduction .....</b>	<b>45</b>
<b>6.2 Testing Process .....</b>	<b>45</b>
<b>6.2.1 Unit Testing .....</b>	<b>46</b>
<b>6.2.1.1 Launch.java .....</b>	<b>46</b>
<b>6.2.1.2 MobileAgentGui.java .....</b>	<b>47</b>
<b>6.2.1.3 AfterClone() of MobileAgent.java .....</b>	<b>47</b>
<b>6.2.1.4 OnGuiEvent() of MobileAgent.java .....</b>	<b>48</b>
<b>6.2.1.5 ReceiveMsg of MobileAgent.java .....</b>	<b>49</b>
<b>6.2.1.6 PersonalDataGui.java .....</b>	<b>50</b>
<b>6.2.1.7 PersonalData.java .....</b>	<b>51</b>
<b>6.2.1.8 Criteria.java .....</b>	<b>52</b>

<b>6.2.2 Component Testing.....</b>	<b>53</b>
<b>6.2.2.1 MobileAgent.java.....</b>	<b>53</b>
<b>6.2.2.2 RegisterAgent.java.....</b>	<b>54</b>
<b>6.2.2.3 QueryEngine.java.....</b>	<b>55</b>
<b>6.2.3 Integration Testing.....</b>	<b>56</b>
<b>6.2.3.1 Integration of Directory and RegisterAgent.....</b>	<b>56</b>
<b>6.2.3.2 Integration of Directory, RegisterAgent and MobileAgent.....</b>	<b>57</b>
<b>6.2.3.3 Integration of Directory, RegisterAgent, MobileAgent and QueryEngine ...</b>	<b>59</b>
<b>6.2.4 White Box Testing.....</b>	<b>60</b>
<b>6.2.5 Black Box Testing.....</b>	<b>60</b>
<b>6.2.5.1 Checking the system on valid data.....</b>	<b>60</b>
<b>6.2.5.2 Checking the system on invalid data.....</b>	<b>61</b>
<b>6.2.5.2.1 Skipping the noncompulsory fields.....</b>	<b>61</b>
<b>6.2.5.2.2 Skipping the compulsory fields.....</b>	<b>61</b>
<b>6.2.6 Static Analysis of Code.....</b>	<b>61</b>
<b>6.2.6.1 Control Flow Analysis.....</b>	<b>62</b>
<b>6.2.6.2 Data Analysis.....</b>	<b>62</b>
<b>6.2.6.3 Interface Analysis.....</b>	<b>62</b>
<b>6.2.7 Conclusion.....</b>	<b>62</b>
<b>7. FUTURE WORK AND CONCLUSION.....</b>	<b>64</b>
<b>7.1 Future Work.....</b>	<b>64</b>
<b>7.2 Conclusion.....</b>	<b>64</b>
<b>APPENDIX A.....</b>	<b>65</b>

<b>APPENDIX B .....</b>	<b>69</b>
<b>APPENDIX C .....</b>	<b>71</b>
<b>APPENDIX D .....</b>	<b>77</b>
<b>APPENDIX E .....</b>	<b>80</b>
<b>APPENDIX F .....</b>	<b>82</b>

## LIST OF TABLES

Table	Page Number
6-1 Test case for Launch.java .....	46
6-2 Test case for MobileAgentGui.java.....	47
6-3 Test case for AfterClone() .....	48
6-4 Test case for OnGuiEvent() .....	49
6-5 Test Case for ReceiveMsg .....	50
6-6 Test Case for PersonalDataGui.java.....	51
6-7 Test Case for PersonalData.java .....	51
6-8 Test Case for Criteria.java .....	52
6-9 Test Case for MobileAgent.java .....	53
6-10 Test Case for RegisterAgent.java.....	54
6-11 Test Case for QueryEngine.java .....	55
6-12 Test Case for Integrated RegisterAgent and Directory .....	57
6-13 Test Case for Integrated MobileAgent, RegisterAgent and Directory .....	58
6-14 Test Case for Integrated MobileAgent, RegisterAgent, Directory, QueryEngine .....	59

## LIST OF FIGURES

Figure	Page Number
3-1 Usecase Diagram .....	19
3-2 Domain Model.....	20
4-1 Architectural Diagram .....	23
4-2 High Level Design .....	24
4-3 Low Level Design.....	26
4-4 Class Diagram.....	27
4-5 Data Flow Diagram .....	28
4-6 Service Discovery by MobileAgent .....	29
4-7 Registration of Job Companies .....	30
4-8 Getting Data and Querying Ontology.....	31
4-9 Setting MobileAgent.....	31
5-1 MobileAgentGui.....	35
5-2 PersonalDataGui.....	36
5-3 RegisterAgentGui .....	41

## 1 Introduction

### 1.1 Preface:

World Wide Web is the universe of information most of which is for humans' perusal, not for computer programs to manipulate meaningfully. Before the arrival of WWW in 1990`s, seeking information was considered to be a very intricate task as there used to be no central resource of information. But with the arrival of internet and the evolution of World Wide Web, this problem was solved to a considerable extent as the internet escaped from the world of computer science and entered the world as a whole. But the problem now is information overloading [1]. The WWW has become so gigantic that it is hard to find what a person is looking for. Most of the information present on the web is in the form of text. For making the search easy for the users, different search engines have been developed. Google is one of the search engine most widely used today. Google and other search engines similar to it basically use string matching with keywords to find relevant documents.[1] This search is syntax based. To the search engines (and other computer programs) the text has no meaning [1]. Text is just strings of characters. Due to this reason a large amount of irrelevant and inaccurate information is returned to the user. The user then has to go through all the results individually to find the pertinent information, making it a very tedious activity which cannot be afforded in the challenging and demanding world of today.

## **1.2 Project Vision:**

The basic idea behind the project is to make the search on the Web easy and intelligent. Another objective is to provide ease to the user by minimizing his intervention in searching and sorting the desired information. This can only be done by automating the search. As all the advances in the field of science and technology aims at one main purpose i.e. “to provide humans as much convenience as possible”. This project hence is an effort to contribute towards achieving this goal.

## **1.3 Proposed Solution:**

The solution proposed makes use of two technologies i.e. *Semantic Web* and *Mobile Agent* to make the search on World Wide Web meaningful and precise in context of job seeking. Semantic Web is a promising technology that is aimed at putting meanings into the contents of current Web, thus enabling the computer programs to understand the contents and consequently process them intelligently. The main feature of Semantic Web is the existence of metadata that facilitates the meaningful and intelligent search by computer programs. In the suggested solution, a personalized Mobile Agent carries out the search on user’s behalf. Mobile Agents are programs that can autonomously travel across a network and perform tasks on machines that provide agent hosting capability. The use of mobile agent reduces network load as instead of downloading large amount of data for processing, the processing is transferred to where data lies. Mobile agents also introduce concurrency as it can clone itself on various hosts to perform processing at the same time.

## **1.4 Aim of the project:**

The aim of the proposed work is to effectively deal with the scenario where a user, is searching for a job that conforms to his specifications of salary, timings, city or country etc, using a conventional search engine. A major portion of the search results will contain the words he entered in any combination and many websites that do not contain the exact words will not be shown although they have job descriptions matching his preferences. Consequently, the user will be faced with a plethora of irrelevant results.

In the recommended solution, the user instead of searching on World Wide Web himself, he provides his job preferences like domain, salary, timings, city or country etc to a Mobile Software Agent that searches on his behalf. The companies who are offering jobs register their job domains and website's location with a central directory. The Mobile Agent first looks up a directory on the basis of the job domain entered by the user like IT, medicine, management etc. The Mobile Agent as a result of this look up gets a list of relevant websites where it clones itself. These website hosting machines have a supporting environment to allow the arrival of Mobile Agent clones. To enable the Mobile Agent to do the meaningful search, semantics are introduced in the websites through the use of ontology. The mobile agent clone processes the ontology thus the resulting search is not on the basis of matching words but on the basis of matching concepts in user's preferences and job ontology. If the user job preferences match with the company's job offers, the clone drops the user's CV on that website hosting machine and returns the website's URL to its parent on the user's machine which is then opened for the user. In this way, the user is able to get the best possible results without excessive effort.



Both the Semantic Web and Mobile Agent are presently among the most emerging technologies. The proposed solution employing a personalized Mobile Agent designed to make a user' search on Semantic Web optimal and efficient, hence, is an attempt to contribute to the ongoing advancements in these fields.

## **1.5 Organization of Project Report:**

The project report has been drafted carefully deciding the sequence to be followed. After the introduction section, the report incorporates the Literature Review chapter summarizing the text studied before and during the project's execution. Subsequently, the System Analysis chapter comes which includes the major interface, functional and non-functional requirements of the system. It also incorporates the usecase diagram and the domain model of the system. Next is the System Design chapter comprising of the architectural diagram, high and low level design, data flow diagram, class diagram and interaction diagram. Following this the report includes the implementation chapter identifying and elucidating the classes which are implemented. Then is the testing chapter incorporating the testing process employed to test the system and the results that were obtained. The next chapter then discusses the work that can be done in future to further enhance the system and ultimately this chapter wraps the report.

## **2 Literature Review**

### **2.1 Introduction:**

This chapter is an effort to summarize the large material that has been studied throughout the project execution. The literature studied can be mainly divided under two main sections: Agents and Semantic Web. Each heading in this chapter actually is a separate area of study in itself but for the purpose of documenting, the material has been abridged. Going through this section will aid in comprehending the later chapters.

### **2.2 Agents:**

Agent is a program with the following properties such as it responds in a timely fashion to changes in environment, exercises control over its own actions, it is goal-oriented and communicates with other agents including people [2].

### **2.3 Mobile Agent:**

A Mobile Agent is a composition of computer software and data which is able to migrate (move) from one computer to another autonomously and continue its execution on the destination computer [3].

### **2.4 Advantages of using Mobile Agent:**

The Mobile Agent is preferred over the static agent in the project because of the many advantages it possesses over static agents. Some of the major features because of which it is preferred are:

### **2.4.1 Reduce the network load:**

The Mobile Agent moves the computation to the data rather than the data to the computation. It means that instead of downloading the data on its machine, the Mobile Agent moves to that machine where the data lies, carrying with it any information or data needed, ultimately leading to reduction in network load.

### **2.4.2 Overcome network latency:**

The use of Mobile Agent decreases response time in critical real time systems. In real time systems, the user wants response as soon as possible so delay in results cannot be afforded e.g. in hospital applications. So use of Mobile Agent provides the added advantage of overcoming the network latency.

### **2.4.3 Asynchronous and Autonomous execution:**

Mobile devices often rely on expensive or fragile network connections. Tasks can be embedded into mobile agents, which can then be dispatched into the network. After being dispatched, the agents become independent of the process that created them and can operate asynchronously and autonomously. The mobile device can reconnect at a later time to collect the agent.

### **2.4.4 Dynamic Adaptation and Robustness:**

Mobile agents can sense their execution environment and react autonomously to changes. Also it provides the advantage of robustness and fault tolerance. If a host is being shut down, all agents executing on that machine are warned and given time to dispatch and continue their operation on another host in the network.

### **2.4.5 Heterogeneity:**

Mobile agents are generally computer and transport layer independent (dependent on only their execution environments), they provide optimal conditions for seamless system integration [4].

## **2.5 JADE-Tool for Developing Agents:**

JADE (Java Agent Development Environment) is a software Framework fully implemented in Java language. It simplifies the implementation of multi-agent systems. It complies with the FIPA specifications. It has a set of graphical tools that supports the debugging and deployment phases [5].

### **2.5.1 Important Terms of JADE:**

JADE has many important terms related to it which must be defined to be understood. Each running instance of the JADE runtime environment is called a *Container*. The set of active containers is called a *Platform*. As soon as a JADE platform starts, the first container to start is called *Main Container*. All other containers register with it as soon as they start. A main container holds two special agents, *AMS* and *DF*. AMS provides a *naming service* which ensures that each agent in the platform has a unique name. It also represents the authority in the platform (for instance it is possible to create/kill agents on remote containers by requesting that to the AMS). Only one AMS will exist in a single platform. Each agent must register with an AMS in order to get a *valid AID*. DF (Directory Facilitator) provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals [5].

### **2.5.2 Agent Communication:**

One of the most important features that JADE agents provide is the ability to communicate. The messages exchanged by JADE agents have a format specified by the ACL language defined by the FIPA international standard for agent interoperability. This format comprises a number of fields and in particular the sender of the message, the list of receivers, the communicative intention (also called “performative”) indicating what the sender intends to achieve by sending the message [5].

### **2.5.3 Reasons of using JADE:**

JADE is employed in the project because it is distributed free of charge and expiration date and in complete form. The programming language it supports is JAVA. Using java holds advantage because it is an **object oriented language** and **portable** i.e. it can run on different operating systems.

## **2.6 Semantic Web:**

The Semantic Web is an extension of the current web, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. The main aims of Semantic Web is the efficient resource discovery and utilization and bringing structure to the meaningful content of Web pages, creating an environment where software agents can readily carry out sophisticated tasks for users [6].

## **2.7 Semantic Web Technologies:**

The Semantic Web is made through incremental changes, by bringing machine-readable descriptions to the data and documents already on the Web. The main semantic

web technologies in sequence of their evolution are, XMLS (XML Schema), XML, RDF, RDFS (RDF Schema), OWL.

### 2.7.1 XML:

XML stands for EXtensible Markup Language markup language much like HTML. It is designed to describe data but the tags are not predefined. We must define our own tags. It uses a Document Type Definition (DTD) or an XML Schema to describe the data. It follows W3C Recommendation [7].

### 2.7.2 RDF:

RDF stands for Resource Description Framework. RDF is a framework for describing resources on the web. It provides a model for data, and syntax so that independent parties can exchange and use it. It is designed to be read and understood by computers. RDF is not designed for being displayed to people. It is written in XML. It is a part of the W3C's Semantic Web Activity and is a W3C Recommendation [8].

### 2.7.3 Ontology:

Ontology is a data model that represents a set of concepts within a domain and the relationships between those concepts. It is used to reason about the objects within that domain. Ontology is used in artificial intelligence, the semantic web, software engineering, biomedical informatics and information architecture as a form of knowledge representation about the world or some part of it.

Ontology generally describes, **individuals** which are basic or "ground level" objects, **classes** that are sets, collections, or types of objects, **attributes** are properties, features, characteristics, or parameters that objects can have and share, **relations** are

ways that objects can be related to one another and **events** which can be perceived as the changing of attributes or relations.

Ontology is now-a-days progressing at a great pace and finds applications in many fields e.g. Artificial Intelligence, Semantic Web Software Engineering and Biomedical informatics [9].

#### **2.7.4 OWL:**

The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML and RDF by providing additional vocabulary along with a formal semantics [10].

##### **2.7.4.1 Reasons of using OWL:**

OWL has been used in the project because of many reasons such as using OWL will enable the Semantic Web to be built on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. It is the first level above RDF required for the Semantic Web which can formally describe the meaning of terminology used in Web documents. OWL has been designed in a way enabling useful reasoning tasks on the Web documents going beyond the basic semantics of RDF Schema [10].

#### **2.7.5 Protégé:**

Protégé is an extensible, platform-independent environment for creating and editing ontology and knowledge bases. It has a friendly user interface enabling the users to conveniently build and edit ontology [11].

### **2.7.5.1 Reasons of using Protégé:**

There are several features that distinguish Protégé from other knowledge base editing tools. No other tool except Protégé has all of the features such as it is intuitive and easy-to-use graphical user interface, its database back-end loads frames only on demand and uses caching to free up memory when needed, and it can easily be extended with plug-ins tailored for user's domain and task [11].

### **2.7.6 JENA:**

Jena is the interface which will be used by the agents to find out whether a certain job website fits to the user criteria by querying its ontology. It is a toolkit for developing applications within the semantic web. Jena provides two main APIs, Jena RDF API and Jena 2 Ontology API [12].

### **2.7.7 Pellet:**

Pellet is a complete OWL-DL reasoner acceptable to very good performance, extensive middleware, and a number of unique features. It is the first sound and complete OWL-DL reasoner with extensive support for reasoning with individuals (including nominal support and conjunctive query), user-defined data types, and debugging support for ontology. It implements several extensions to OWL-DL including combination formalism for OWL-DL ontology, a non-monotonic operator, and preliminary support for OWL/Rule hybrid reasoning [13]. Pellet is written in Java and is open source.



### **2.7.7.1 Difference between Pellet and JENA Query Engines:**

The Jena query engine is RDF triple based, so to produce variable bindings it works one triple at a time. In contrast, the Pellet query engine considers the entire conjunctive query. This difference causes the engines to have different performance characteristics and in some cases yields different results.

First, by using a level of representation consistent with the logic, the Pellet query engine can perform optimizations that are inaccessible to the Jena query engine. These optimizations often yield a significant speed-up in query answering.

Second, results may differ based on handling of blank nodes in queries. The semantics of SPARQL are such that blank nodes need not be bound to asserted individuals and can match individuals that are inferred (such as those from existential restrictions and minimum cardinality constraints). The Pellet engine will produce results using these inferred individuals and the Jena engine will not [14].

### **2.7.7.2 Using Pellet in JAVA Applications:**

Pellet can be accessed from Java applications using one of two different APIs designed to support **Jena** and **OWL API** libraries [15]. In the project Jena libraries are being used.

### **2.7.7.3 Using Pellet with JENA:**

There are two different ways to use Pellet in a Jena program: either using direct Pellet interface which is highly recommended or using Jena DIG interface which is not recommended. The Direct Pellet interface is much more efficient (e.g. does not have the HTTP communication overhead) and provides more inferences (DIG protocol has some

limitations). Using the direct interface is not any different than any other Jena reasoner [15]. The project employs the direct Pellet interface.

## **2.8 Conclusion:**

This chapter incorporated the explanation of all the significant technical terms which are used in the fore coming chapters. It also elucidated the alternative technologies that might have been considered as an option to be used in the project implementation but the comparison done in this chapter has made it clear that why one technology is preferred over the other. Hence this chapter gives a comprehensive idea about the technologies being used in the project which will enable the better understanding of the subsequent chapters.

### **3 System Analysis**

#### **3.1 Introduction:**

This chapter covers the system analysis phase of the project. In this phase, first of all the scope of the project is presented as it's clear definition and understanding is needed for the absolute comprehension of the system's requirements. After the scope, the requirements' specification phase, including major functional and non-functional requirements, is described. The requirements' specification phase is then followed by usecase diagram and domain model of the system for the better understanding of the system analysis phase of the project.

#### **3.2 Project Scope:**

The aim of the project is to make the search in the context of job seeking meaningful and optimal by implementing a scenario where a user wants to search for a job which matches his specifications of domain, city, salary, timing hours etc The user has a personalized Mobile Software Agent which takes these job preferences and personal data from him and carry out the search on his behalf. A central directory is there with which the companies offering jobs register their job domain and website's location. The companies have put semantics in their websites thus enabling the Mobile Agent to process them intelligently. The Mobile Agent looks up the directory to short list the websites in the basis of job domain. It then clones to the selected websites where it queries their semantic documents to determine whether the user's job specifications match with the company's job offers. If they do, then the clone returns the URL of the

website to its parent which opens it for the user. The clone also drops the user's CV on the website hosting machine. This search is not on the basis of matching strings as is done in conventional search engines. Rather it is based on the matching of concepts that are specified in user's preferences and websites' semantic documents. The user hence instead of facing a plethora of irrelevant information, gets the most optimal and precise results.

### **3.3 Requirements Specification:**

The requirements specification of the system includes its external interface requirements including user and software interface requirements. It also involves the analysis of the major functional requirements including the main functionalities that the system is expected to deliver. Analysis of main non-functional requirements also is important as it tends to give an idea about the characteristics of the system such as accuracy, scalability etc.

#### **3.3.1 External Interface Requirements:**

It means requirements which include the interaction of the system with the external requirement e.g. user interface through which the user interacts with the system and software interfaces means the software tools which are employed in the system.

##### **3.3.1.1 User Interfaces:**

Several GUIs are required to enable the user and the job companies to interact with the system for example; a GUI to get the job preferences from the user, another to take the personal data from the user and another GUI is needed to facilitate the job companies to register with the directory.

### **3.3.1.2 Software Interfaces:**

The project employs Windows XP as the operating system and software tools such as, JADE, Pellet and Protégé. JADE is used to provide an agent platform, Pellet for querying the ontology and Protégé for writing the ontology.

### **3.3.2 Major Functional Requirements:**

Functional requirements means the core functionalities that the system is meant to deliver, e.g. acquiring user's data, creating semantic websites, registration of websites with the directory, directory look up by Mobile Agent, cloning to related websites, processing the ontology, returning the result and dropping the user's CV. These functionalities are illustrated in more details under the following headings.

#### **3.3.2.1 Acquiring User's Preferences and Personal Data:**

The user provides his job priorities to his personalized Mobile Agent. The preferences include the field in which he is interested to find a job, job title, salary, timings and location which includes both city and country. He will also specify his personal data that the job companies often require which includes applicant's full name, age, experience and qualification.

#### **3.3.2.2 Creating Semantic Websites:**

To enable the Mobile Agent to make a meaningful and an intelligent search, semantic documents need to be created for the websites, describing the concepts specified in them and forming relationships between those concepts

### **3.3.2.3 Registration of Websites with Directory:**

The job offering companies register their job domains, sub domains, websites' index pages' names and ontology's names and the websites' locations with a directory so that they are available for the search by Mobile Agent.

### **3.3.2.4 Directory Look Up by Mobile Agent:**

The Mobile Agent looks up the directory to find the websites offering jobs in the domain specified by the user. It then acquires the locations of the relevant websites from the directory.

### **3.3.2.5 Cloning to the Relevant Websites:**

The Mobile Agent on acquiring the locations of the relevant websites sends its clone to each of these locations.

### **3.3.2.6 Processing of Semantic Websites by Mobile Agent Clone:**

The Mobile Agent clone will start processing those websites to compare the concepts specified in them with those provided by the user to see whether the jobs specified in those websites match user's preferences or not.

### **3.3.2.7 Returning the results:**

The mobile agent clone after processing the relevant websites returns the results to the user. The result can be returning the webpage's URL in case the user's data matches with the job offers or just a message displaying "Sorry, no results are found" in case they don't match.

### **3.3.2.8 Dropping the user's CV:**

If the job preferences of the user match with the job offer of the company, the Mobile Agent drops the CV of the user on the website hosting machine. The CV is dropped only if the user provides the Mobile Agent with his CV initially while providing his personal data. Otherwise it just returns the URL of the web page to the user.

### **3.3.3 Major Non-Functional Requirements:**

The main non-functional requirements that the system provides are accuracy and user friendly GUIs. Accuracy is there because semantic web is being used and the major advantage that it provides over World Wide Web is the precision of results. GUIs should be welcoming so that the user finds it easy to understand.

#### **3.3.3.1 Accuracy:**

The search performed by the agent should provide very accurate and precise results to the user as the search will be semantic based instead of string matching. The number of results will be smaller in number but exact, enabling the user to get rid of huge bundle of inaccurate results.

#### **3.3.3.2 User friendly GUI:**

The user interface for getting the job preferences and personal data from the user should be friendly and interactive. Also the GUI used by the companies to register their job details should be pleasant as well.

### 3.4 Usecase Diagram:

The usecase diagram of the system is shown in Figure 3.1. From the figure, it can be seen that there are three primary actors involved in the system which are, Mobile Agent, Mobile Agent Clone and Job Company. The main jobs of the Mobile Agent (i.e. its usecases) are that it acquires user's job preferences and personal data, looks up the directory, clones to relevant websites. The usecases for Mobile Agent Clone are to process semantic websites, return the results and dropping user's CV. The Job company has usecases such as to create semantic websites and register websites with the directory.

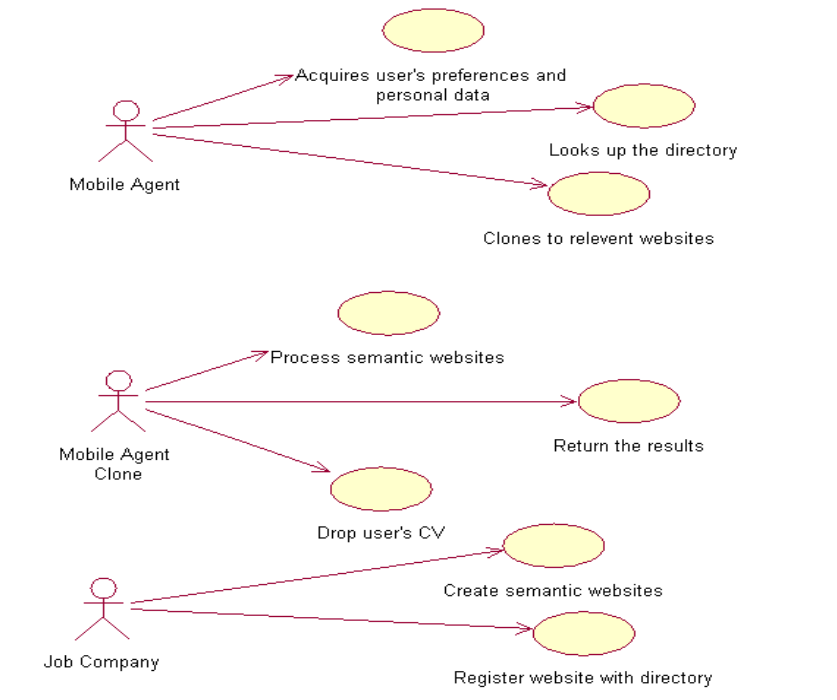


Figure 3-1: Usecase Diagram

### 3.5 Domain Model:

The domain model of the system is shown in Figure 3.2. The main concepts which have been identified are: User, Job preferences, Personal data, CV, Mobile Agent, Mobile Agent clone, Directory, Website, Website host, Query engine, Job, Job requirements, Job



characteristics, Job Company and Ontology. The associations among these concepts have been shown as well making the system easy to comprehend.

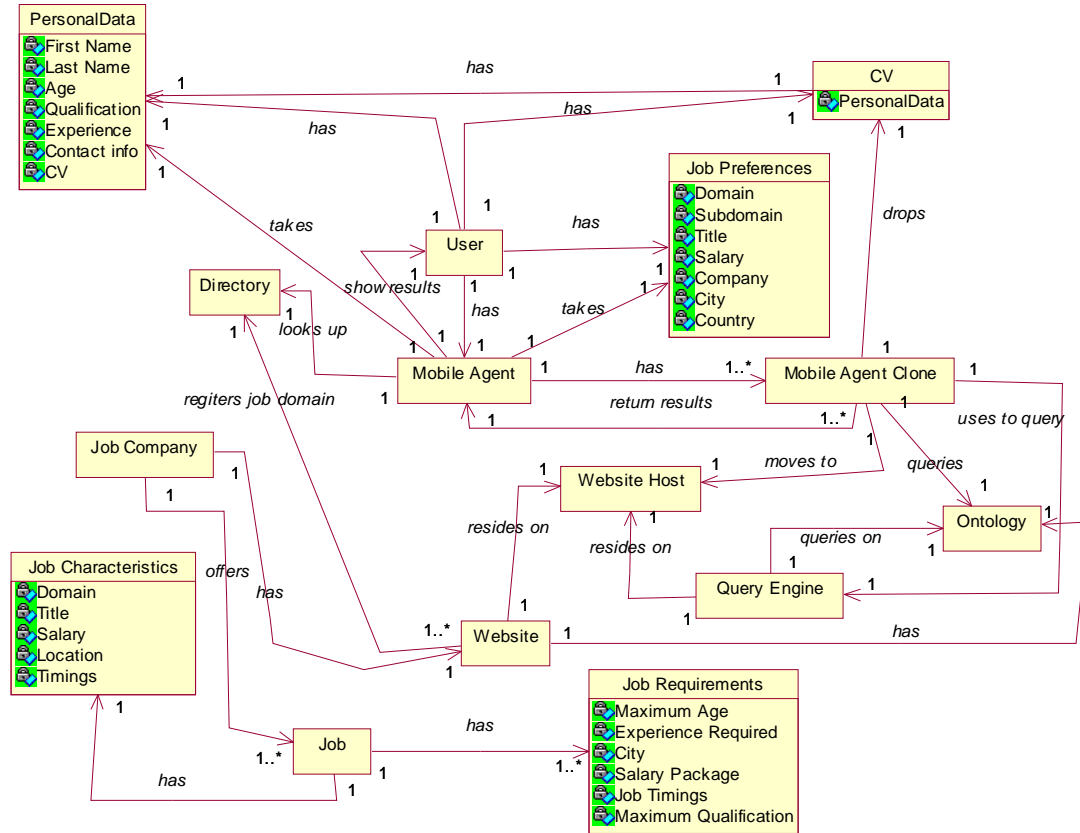


Figure 3-2: Domain Model

### 3.6 Conclusion:

The system analysis of the project has been covered in this chapter. The scope of the project has been revised for the clear understanding of the requirements, key functional and non-functional requirements have been enumerated, the usecase diagram showing the major actors and their actions have been included such as Mobile Agent acquiring the user’s preferences and personal data, cloning to pertinent websites, Mobile Agent Clone processing the websites and returning results and Job company creating and registering

websites. The conceptual model identifying the major concepts in the system has been incorporated as well. This chapter has been written comprehensively so that the fore coming design chapter becomes easy to comprehend.

### 4 System Design

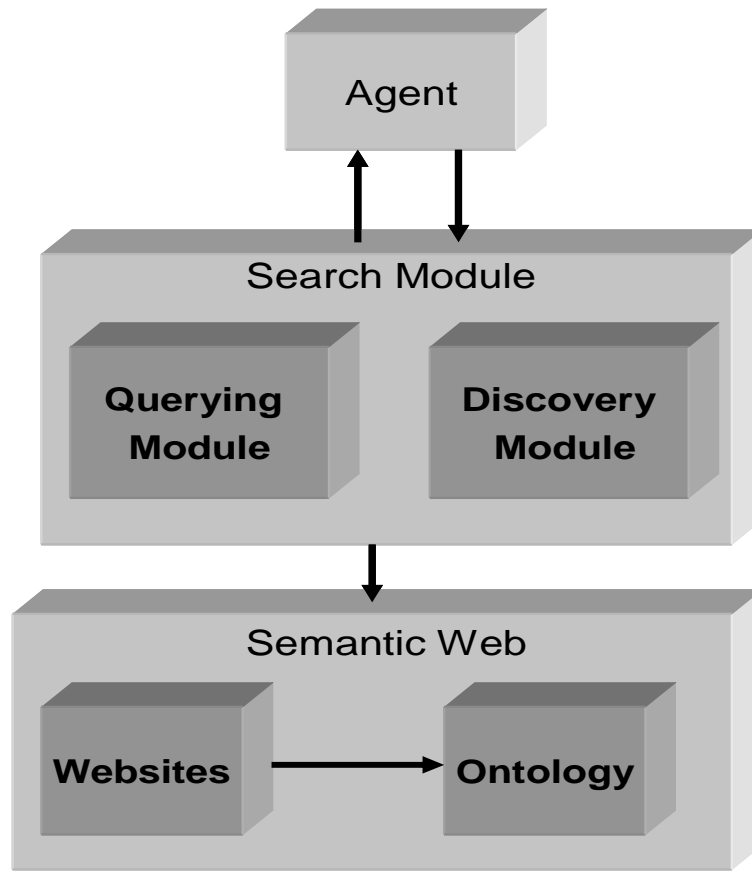
#### 4.1 Introduction:

System design is a very important phase in the software development process. The succeeding implementation phase is performed taking into consideration the design constraints. This chapter begins by presenting the high level design of the project showing the main modules of the system without including much detail. Next the low level design is incorporated elucidating the modules identified in the high level design. It is then followed by the data flow diagram of the project. Class diagram is also included focusing on the implemented classes, their attributes and their relationships with each other.

#### 4.2 Architectural Diagram:

The architectural diagram of the system is specified in Figure 4.1. It illustrates the basic areas of the project which are: Agent, Search module and Semantic web

As depicted in Figure 4.1, the communication between the agent and the search module is bidirectional as the agent first uses the discovery module to find out the websites offering jobs in the same domain and sub domain as specified by the user. Each website has an ontology written for it. The querying module is then used to query the selected website's ontology and the corresponding results are returned to the user.



**Figure 4-1: Architectural Diagram**

### **4.3 High Level Design:**

The high level design of the project is shown in the Figure 4.2. It is built using black box approach, focusing on the main modules of the system and not considering their inner details. The Figure 4.2 identifies the three fundamental modules of the project as the Directory, User's machine and Website Hosting machine.

As can be depicted from Figure 4.2, there are websites which register themselves with a directory. The website hosting machines contain job ontology to provide semantics to the websites, Mobile Agent clone and a supporting environment to allow the clone to come. A mobile agent resides on the user's machine which looks up the directory to find

the websites offering jobs in the domain as specified by the user and then clones to the selected websites.

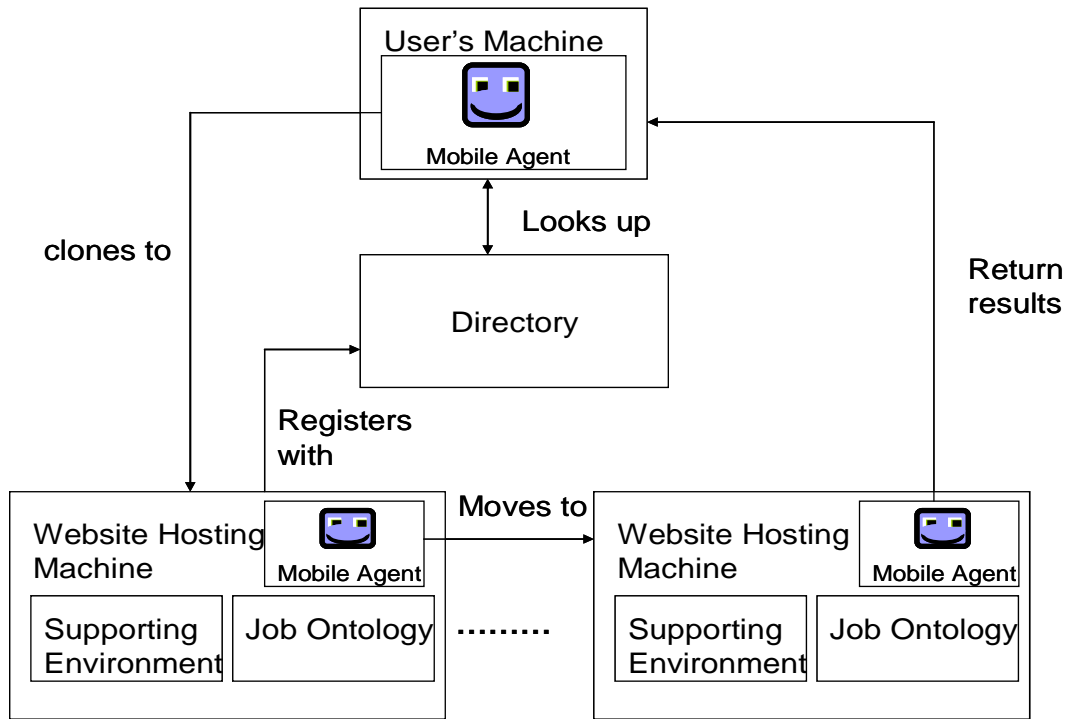


Figure 4-2: High Level Design

#### 4.4 Low Level Design:

The Figure 4.3 illustrates the detailed low level design of the project. Here each module has been explained in more detail. An agent platform has also been employed in the project which has been explicated as well.

##### 4.4.1 Agent Platform:

As the suggested solution is agent-based, so an agent platform is needed to aid the presence of Mobile Agent on all machines. The platform employed in the project is of

JADE (Java Agent Development Environment). It is a software framework which has been fully implemented in JAVA. The agent that is employed from JADE to provide its functionality as a directory is DF (Directory Facilitator) Agent. JADE supports agent mobility within a platform. So presence of JADE is compulsory on all the three modules mentioned. This makes the system platform dependent.

#### **4.4.2 Directory:**

The main container of the JADE runs on this host. The agents of the websites offering jobs register their job domains and locations with the Directory Facilitator (DF) agent of the main container.

#### **4.4.3 User's Machine:**

On the user's machine, the personalized mobile agent of the user runs in a separate container which joins the main container on the directory host, hence making both machines part of the same platform. The Mobile Agent has a graphical user interface which takes the user's job preferences and personal data.

#### **4.4.4 Website Hosting Machine:**

The website and its ontology reside on the Website Hosting Machine. Ontology is written using the ontology editor Protégé. To get registered to the directory, an agent Register Agent runs on the website hosting machine in a container, which then joins the main container running on the directory host. Consequently, the website hosting machine, user's machine and the directory host become part of the same platform.

#### 4.4.5 Search by Mobile Agent:

The Mobile Agent which runs on the user's machine looks up the directory host to get the locations of the registered websites offering jobs in the domain specified by the user. After getting their locations, the Mobile Agent clones itself and each clone moves to one of those locations. The clone on reaching the website host, queries the ontology through pellet. If the job description in the ontology matches the user's preferences, the clone returns the URL of this website to its parent on the user's machine, drops the user's CV on the web hosting machine and terminates itself. The parent Mobile Agent opens this website for the user.

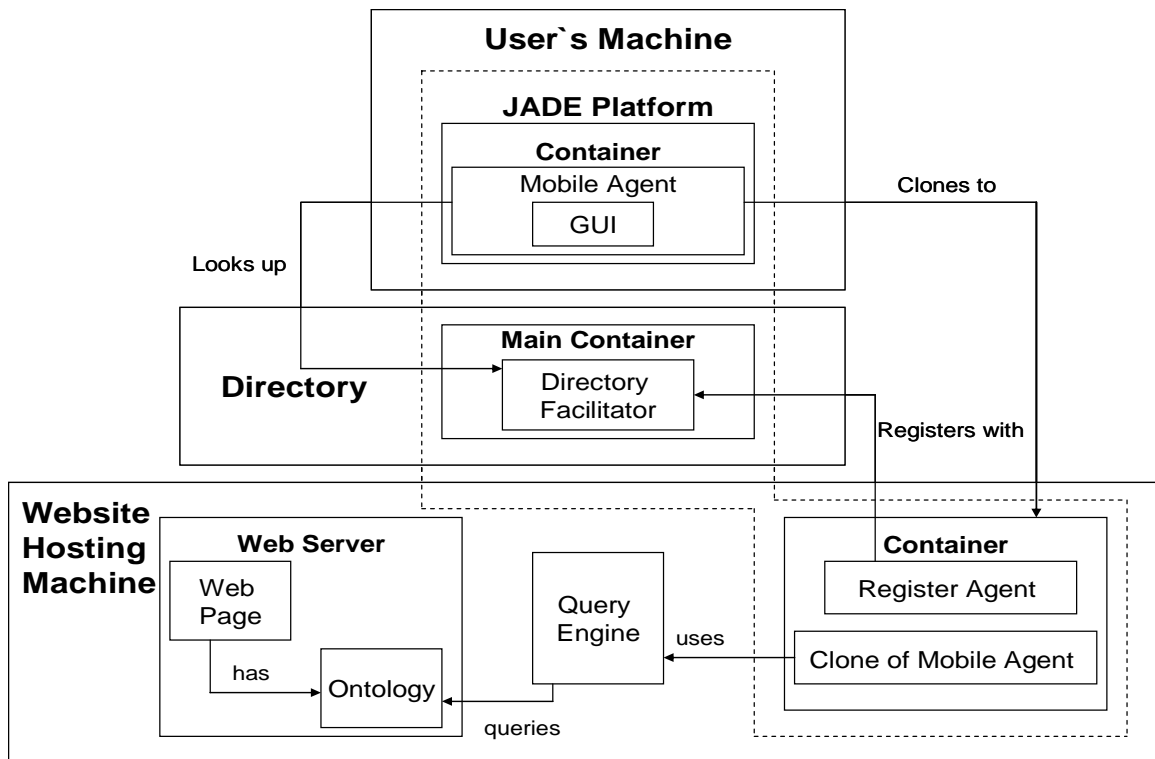


Figure 4-3: Low Level Design

## 4.5 Class Diagram:

Class diagram of the system is illustrated in Figure 4.4. In this diagram the classes that have been implemented are shown along with the inherited classes and the implemented interfaces, along with identification of their relationships with each other e.g. the agent classes MobileAgent and RegisterAgent both inherit from GuiAgent class that extends the Agent class. Similarly the GUI classes like RegisterAgentGui and MobileAgentGui extends the class JFrame and are being instantiated by RegisterAgent and MobileAgent respectively.

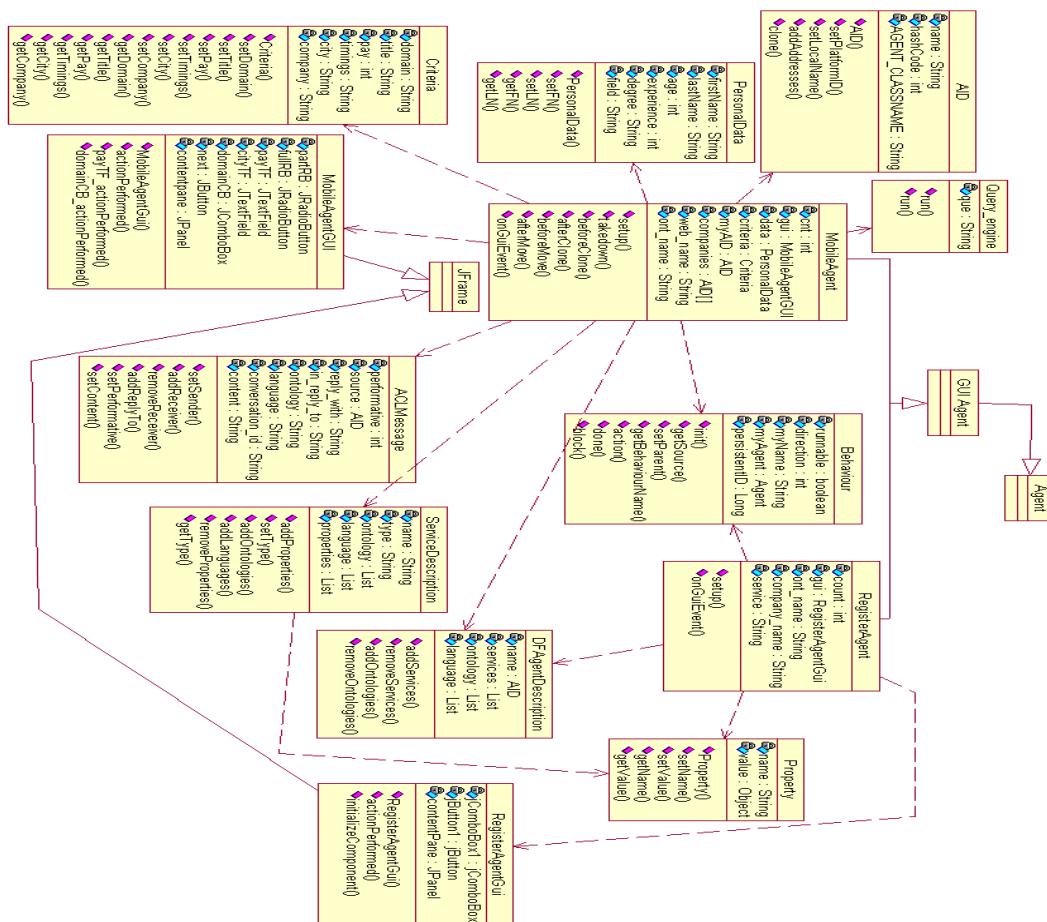


Figure 4-4: Class Diagram



## 4.6 Data Flow Diagram:

The data flow diagram of the system is given in Figure 4.5. First of all the semantic websites register themselves with the directory and the Mobile Agent takes the job preferences from the user. Based on the user's job domain and sub domain specification and the registered websites in the directory, it finds out the websites offering jobs in that particular domain and sub domain as given by the user. On retrieving the locations of such websites, it clones to their hosting machines. The clone on each machine then queries the ontology using Pellet API. The clones then return results to the parent Mobile Agent in the form of web pages' URL.

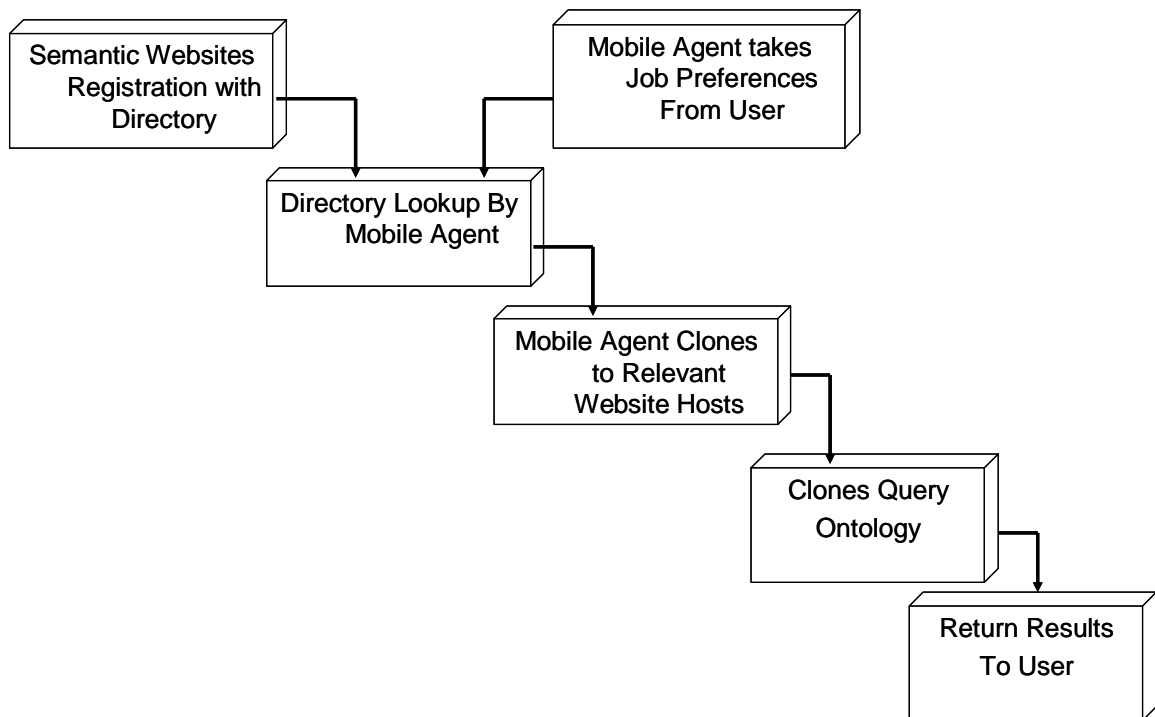
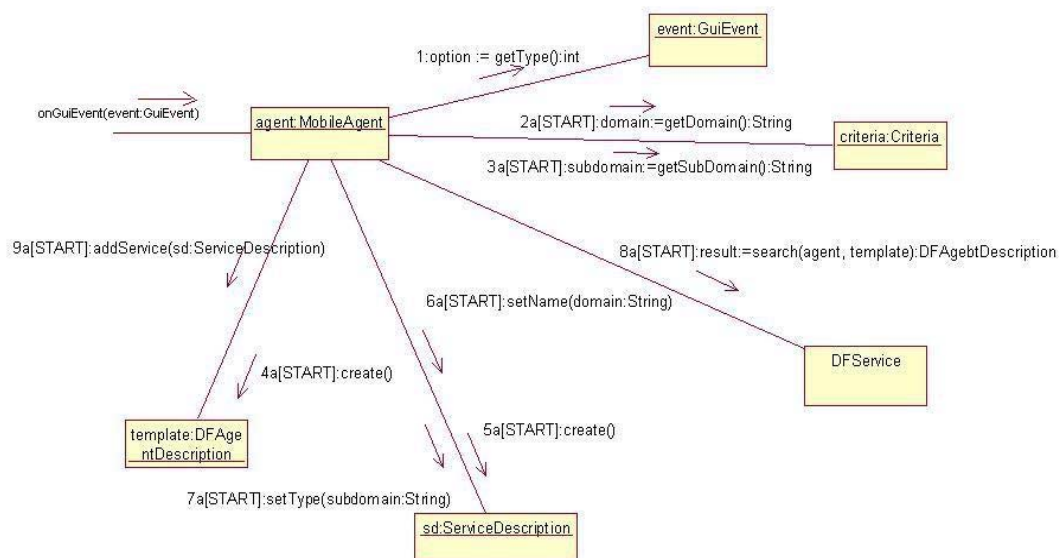


Figure 4-5: Data Flow Diagram

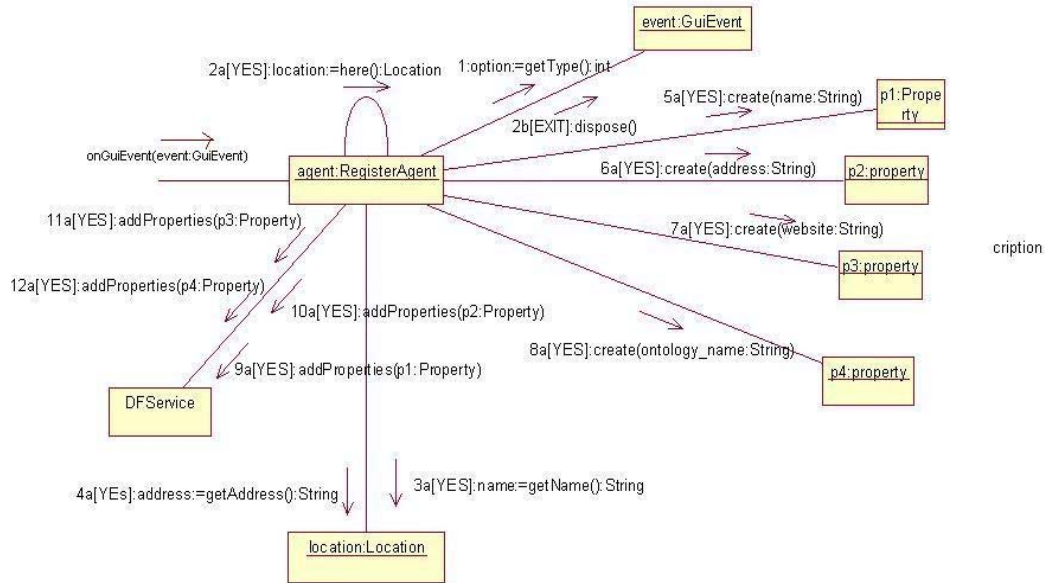
## 4.7 Interaction Diagram:

When job seeker clicks Search button after providing his job criteria and personal data, class *PersonalDataGui* posts an event to *MobileAgent* as a result of which *onGuiEvent()* method in class *MobileAgent* is called. The messages that are called subsequently are shown in the order alongwith the classes on which they are called in Figure 4.6.



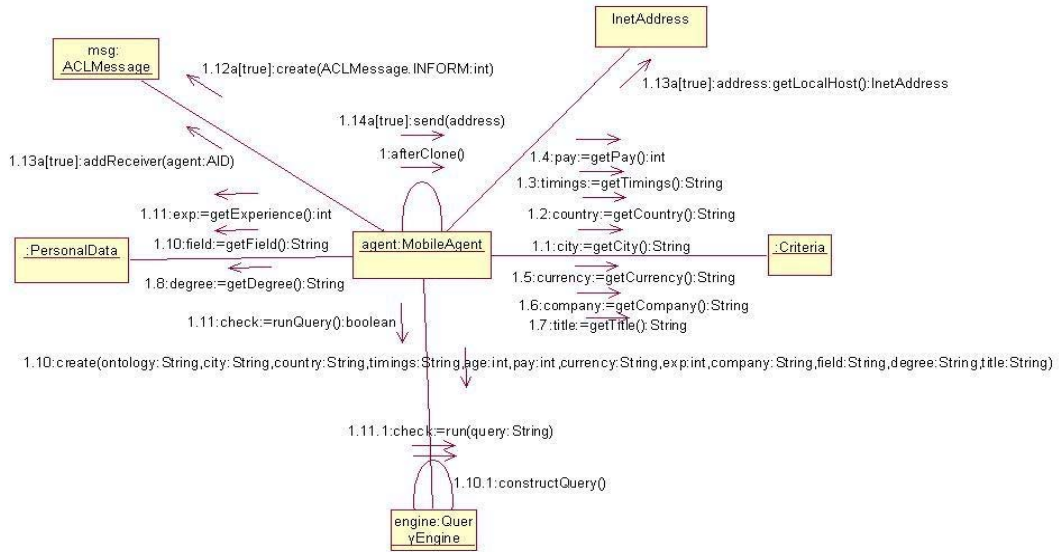
**Figure 4-6: Service Discovery by Mobile Agent**

When a user on the employer’s side clicks Register button to register a job his company is offering, *PersonalDataGui* posts an event to *RegisterAgent* as a result of which *onGuiEvent()* method in class *MobileAgent* is called. The messages that are called subsequently are shown in the order alongwith the classes on which they are called in Figure 4.7.



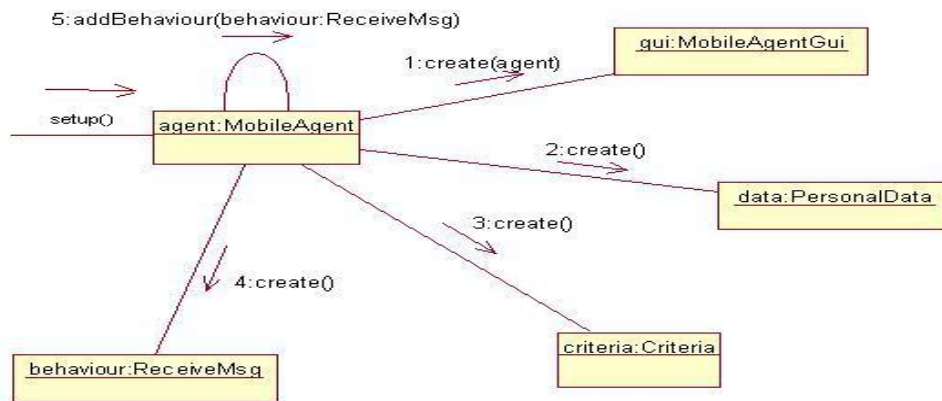
**Figure 4-7: Registration of Job companies**

When Mobile Agent clones, its `afterClone()` method is called in which all actions to be taken at the website host are carried out. Those actions include retrieving the criteria and personal data provided by job seeker and then executing methods of class *QueryEngine* to query the ontology. If user's provided data match with the ontology data, true is returned by *QueryEngine* as a result of which clone of Mobile Agent sends the website address to Mobile Agent. Otherwise clone sends a failure message to Mobile Agent. This is illustrated by interaction diagram in Figure 4.8.



**Figure 4-8: Getting Data and Querying Ontology**

When Mobile Agent is launched, its `setup()` method is called in which `MobileAgentGui` is instantiated that displays gui for the user to enter his job criteria. Class `PersonalData` for setting and getting personal data of user and class `Criteria` for getting and setting user's job criteria are also instantiated. Mobile Agent also receives messages from its clones. This reception of messages is handled by class `ReceiveMsg` which is added to agent's behaviours. This is illustrated by the interaction diagram in Figure 4.9.



**Figure 4-9: Setting MobileAgent**

## **4.8 Conclusion:**

This chapter presented the architecture for Personalized Mobile Agent based search on Semantic Web. It has incorporated the high level design, low level design, data flow diagram and class diagram of the system. Three main modules have been identified which are Directory, User's machine and Website hosting machine. The design is platform dependent as the presence of JADE is mandatory on all the modules of the system. The results returned by the Mobile Agent are going to be more precise as searching on Semantic Web is done on the basis of matching concepts contrary to matching strings as in the conventional search engines. The system design chapter included details that are very important in comprehending well the upcoming implementation chapter.

### **5 Implementation**

#### **5.1 Introduction:**

This chapter presents the implementation details of the project. The coding is done in the object oriented language JAVA. As illustrated in the prior chapter, there are three modules of the system, User's machine, Directory and Website Hosting machine. As the system is distributed, so the classes which are implemented are disseminated among these modules. The implementation chapter is hence structured in the same way i.e. mentioning each module and then elucidating the classes which are needed on that module.

#### **5.2 Implementation language:**

The implementation language which has been chosen for the project is JAVA. It has been preferred over other languages because of several reasons e.g. it is a platform independent language which enhances the system's portability. This factor is significant as the system built is of distributed nature. Also it is an object oriented language which makes it simple to visualize the objects in real life. The agent platform which the system has employed is of JADE which is also fully implemented in JAVA. JADE is used due to the fact that it is distributed free of expiration date and full support is provided to its users in the form of documentation. The Pellet API which has been incorporated in the system to query ontology has also been implemented in JAVA.

### **5.3 Distribution of classes with respect to Modules:**

Considering in account the fact that the system is distributed, the implemented classes will be explained with respect to their placement in the system. From the high level design in the preceding chapter, the three identified modules are User's machine, Directory and Website Hosting machine.

Each module of the system contains a certain set of implemented JAVA classes and their interaction carry out the operation for which they are destined for i.e. job seeking facility for the user on semantic web.

#### **5.3.1 User's machine:**

This component of the system holds a set of classes which collectively form the User Interface. It means that it is through these classes that the user interacts with the system without indulging in the internal details of the system. The classes present on this module can mainly be separated into two sets.

First set contains classes which provide the Graphical User Interface. This group includes two classes which are: MobileAgentGui and PersonalDataGui.

The other set comprises those classes which carry out the other functionality behind these GUI classes. These classes include Launch, MobileAgent, PersonalData and Criteria.

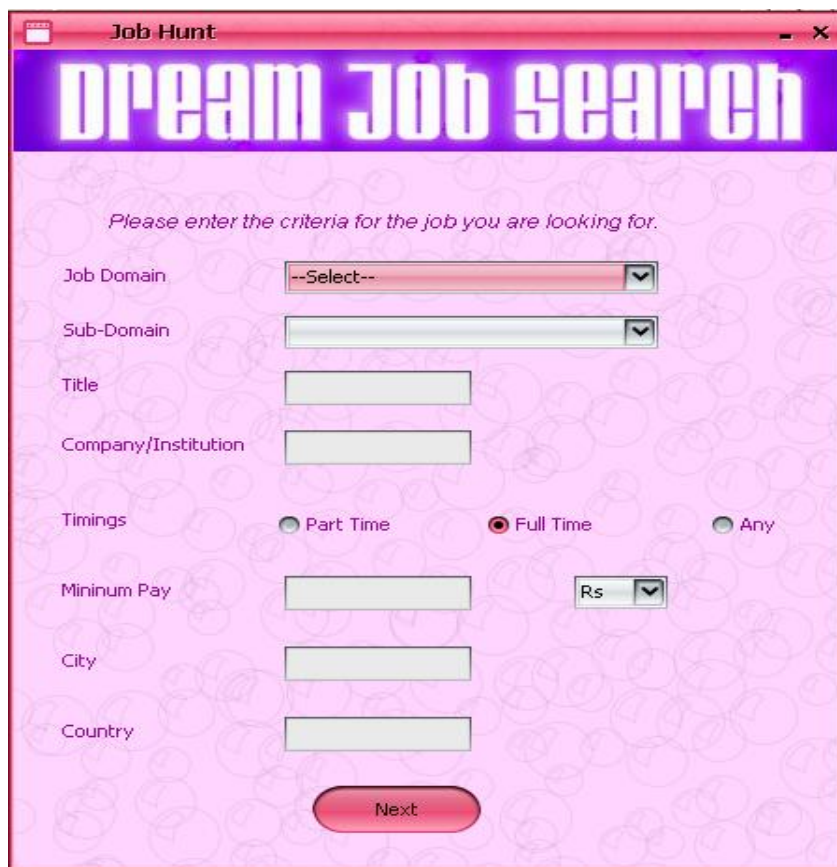
##### **5.3.1.1 Initiating Mobile Agent:**

It is done by class Launch.java. The user instead of setting the class path himself to begin the Mobile Agent he just runs this class as a result of which the Mobile Agent is launched and its GUI appears in front of the user.

### 5.3.1.2 Job Seeker GUI for Acquiring his Job Criteria:

The user interacts with his personalized Mobile Agent through the interface class `MobileAgentGui.java`. It inherits the GUI characteristics from class `JFrame`. The output of its code execution is a GUI form shown in Figure 5.1, which has fields where user enters his job preferences so that the Mobile Agent can carry out the respective search. The job specifications include domain, sub domain, title, company, salary, job timings, city and country as can be seen from Figure 5.1.

As JADE platform is employed to support the agent mobility, so this class imports some of the JADE inbuilt classes and JAVA classes from the package `javax.swing` for the GUI and `java.awt` and `java.awt.event` for action listening.



The screenshot shows a window titled "Job Hunt" with a purple header bar containing the text "DREAM JOB SEARCH" in white. Below the header, there is a pink background with a pattern of faint circles. The text "Please enter the criteria for the job you are looking for." is displayed in a smaller font. The form contains the following fields and controls:

- Job Domain: A dropdown menu with "--Select--" as the selected option.
- Sub-Domain: A dropdown menu.
- Title: A text input field.
- Company/Institution: A text input field.
- Timings: Three radio buttons labeled "Part Time", "Full Time", and "Any". The "Full Time" radio button is selected.
- Minimum Pay: A text input field followed by a dropdown menu with "Rs" as the selected option.
- City: A text input field.
- Country: A text input field.
- Next: A red button with the text "Next" in white.

Figure 5-1: MobileAgentGui



### 5.3.1.3 Job Seeker GUI for Acquiring his Personal Data:

PersonalDataGui.java is a Graphical User Interface class which outputs a form in which the user enters his personal data that includes his first name, last name, age, qualification, experience, contact information and curriculum vitae. The form is shown in Figure 5.2. A text area is provided where the user pastes his CV so that the Mobile Agent clone can drop it in the form of a MS word document on the Website Hosting machine if the user's job preferences match with the job offer of the company. The user's personal data is required to match it with the job company's requirements e.g. qualification needed, experience required etc. This class extends JFrame to make use of its GUI capabilities. It imports the same JAVA and JADE classes as the MobileAgentGui does.



The screenshot shows a Java Swing window titled "Personal Data". The window has a red header bar with the text "DREAM JOB SEARCH" in white, bold, uppercase letters. Below the header, the form is set against a light yellow background with a subtle grid pattern. The form contains the following fields and controls:

- First Name**: A text input field.
- Last Name**: A text input field.
- Age**: A spin box followed by the text "years".
- Qualification Degree**: A dropdown menu with "--Select--" as the selected option.
- Field**: A text input field.
- Experience**: A spin box followed by the text "years".
- Contact Info**: A text input field.
- Paste CV**: A large, empty text area.
- Search**: A rounded rectangular button at the bottom center of the form.

Figure 5-2: PersonalDataGui

#### **5.3.1.4 Manipulating Personal Data:**

The JAVA class which provides the functionality to PersonalDataGui class is PersonalData.java. It uses the interface Serializable. All the information that is entered by the user in the form output by PersonalDataGui class is assigned to its private data members which are firstName, lastName, age, experience, degree, field and CV. This class provides getters and setters to access these data members for manipulation. It imports only one package i.e. java.io to utilize the interface Serializable.

#### **5.3.1.5 Manipulating Job Criteria:**

The job preferences entered by the user in the MobileAgentGui are assigned to the private data members of Criteria.java. These members are named as domain, subdomain, title, company, pay, timings (include part time or full time), city and country. All data members are of type String except pay which is of type integer. The getters and setters required for the manipulation of data as user's job preferences are provided by this class.

#### **5.3.1.6 User's Personalized Mobile Agent:**

The MobileAgent.java is a very important class on the user's machine module. It extends the GuiAgent class and imports a number of JADE and JAVA packages. It instantiates PersonalData and Criteria as its data members both of which are used by classes PersonalDataGui and MobileAgentGui for assigning the values entered by the user in their forms to the data members of PersonalData.java and Criteria.java respectively. This is made possible by sending the reference of the MobileAgent class to the constructors of the GUI classes PersonalDataGui.java and MobileAgentGui.java. As the user runs the MobileAgent class according to the manual's instructions provided to

him, a MobileAgentGui form appears in which the user enters his job preferences. After pressing the Next button provided in the form, a PersonalDataGui form appears in front of him where he enters his personal data and pastes his CV in the specified text area. The MobileAgent then looks up the Directory Facilitator Agent provided by JADE where the websites have registered their job domains, location, ontology name and the website's home page name. The MobileAgent using the JADE classes, **DFAgentDescription** and **ServiceDescription** will locate the websites offering jobs in the same domain as preferred by the user. It then clones to the selected website hosting machines and the clones query the ontology residing on those machines using pellet API. If the job ontology matches with the job preferences of the user, the clone writes the CV on a MS word document and drops it on the website hosting machine. It also finds the URL of the website and sends it to the parent Mobile Agent through as ACL message, which opens it for the user on user's machine. The Mobile Agent clone then terminates itself using method doDelete () from the JADE API.

### **5.3.1.7 Query Engine:**

The class QueryEngine.java uses Pellet API to query the ontology on the website hosting machine to determine that whether the user's job preferences match with the job offers of the company. There is only one private data member in this class which is **que** of type String to store the SPARQL query. The constructor of Query Engine class takes User Preferences and Personal Data as arguments and then the SPARQL Query is constructed within the body of constructor according to the data provided in the arguments. It has two public functions which are run (String queryStr) and runQuery ().

#### **5.3.1.7.1 run (String queryStr):**

It has the return type of boolean and it takes the SPARQL query as an argument. It then creates an empty ontology model using Pellet specification. The code then reads the ontology from the URI specified in the SPARQL Query and loads it in the empty ontology model. An object of an inbuilt Query Execution class is instantiated and its method **execAsk ()** is called to query the ontology model using a SPARQL Ask Query. At the end the result of the Ask Query is returned to its caller function run.

#### **5.3.1.7.2 runQuery ():**

This function also returns a value of type boolean. It calls the method **run ()** passing a SPARQL query as an argument. The result is returned by the run () to its caller runQuery () which ultimately returns it to its own caller function which is in fact the main () of the JAVA code.

### **5.3.2 Directory:**

The main container of the JADE platform runs on this machine. The main container has a special agent Directory Facilitator (DF) agent which serves the purpose of the directory. The Mobile Agent running on user's machine get registered with this DF agent as soon as it starts. The websites register their job domain, location, website homepage name and the ontology name with this DF agent through an agent called **RegisterAgent**. This agent runs on the websites' hosting machines.

### **5.3.3 Website Hosting machine:**

Three classes reside on this module of the system which are RegisterAgentGui, RegisterAgent and Ontology.

The first two classes provide the registration facility to the job companies through which they get registered with the Directory Facilitator agent running on the directory. The Ontology is queried by the Mobile Agent to find that whether the job preferences of the user match with the job offers of the company.

#### **5.3.3.1 Initiating Register Agent:**

It is the class that initiates the RegisterAgent. The companies instead of setting the class path themselves to begin the RegisterAgent, they just run this class as a result of which the RegisterAgent is launched and its GUI appears through which the companies can register their job descriptions with the directory.

#### **5.3.3.2 Job Employers GUI for Job Registration:**

This class RegisterAgentGui.java is instantiated by RegisterAgent.java and provides a friendly Graphical User Interface to the job companies enabling them to register their job domain, sub domain, website index page name and ontology name with the Directory Facilitator agent running on the directory machine. As can be seen from Figure 5.3, the domain is selected from a drop down list provided and then from the corresponding sub domains, the sub domain can be selected using the drop down list. The website index page name and the ontology name are also registered considering the possibility that there can be more than one websites being hosted on a single machine. The RegisterAgent passes its reference in the RegisterAgentGui class's constructor. The

data entered into the GUI is assigned to the data members of the RegisterAgent using its reference. The GUI has a Register button which invokes an action listener on being pressed, hence registering the data entered into the GUI with the directory.



**Figure 5-3: RegisterAgentGui**

### **5.3.3.3 Registration of Jobs:**

RegisterAgent.java is a very significant code among all which reside on the website host. The job companies run this code to register their job domains, sub domains, locations, name of the website index page and the ontology with the yellow pages catalogue managed by the Directory Facilitator agent so that the Mobile Agent can dynamically discover them. Its major data members include RegisterAgentGui, domain, subdomain, ont\_name and webpage\_name.

The data entered by the job companies into the RegisterAgentGui is assigned to these data members. The RegisterAgent discovers its location in the form of the name and address of the object of type **Location** obtained by method here (). The location, webpage name and ontology name are assigned to objects of type **Property**. A **ServiceDescription** object is instantiated and its name is set to be the job domain name and its type is set to be the job sub domain. Four Property objects (location, webpage name and ontology name) are added to this ServiceDescription object. Then **DFAgentDescription** object is instantiated and through the method addService () of the DFAgentDescription, the ServiceDescription object is added to the DFAgentDescription object. This DFAgentDescription object can then be discovered by the Mobile Agent if its name and type matches the domain and sub domain entered by the user in the MobileAgentGui.

#### **5.3.3.4 Ontology:**

The ontology written includes two OWL files, jobs.owl and your\_ontology.owl. The jobs.owl contains a three level of class hierarchy which is illustrated below:

##### **5.3.3.4.1 First Level:**

On first level, there is only one owl class labeled as Occupation. This class has only one data type property (attribute) i.e. synonyms of type String.

##### **5.3.3.4.2 Second Level:**

On second level of hierarchy, there are 15 owl classes which represent the job domains. Each class at this level inherits synonyms property from Occupation class.

#### **5.3.3.4.3 Third Level:**

On third level, there are many classes and their instances are used as job titles. Each class at this level also inherits synonyms property from its super class.

your\_ontology.owl contains a class named as Job which has seven attributes as company, location, pay, qualification, title, age and experience. Job's attributes are all classes as well.

#### **5.3.3.4.4 company:**

It has two attributes, one is company\_name which is data type property of type String and the other one is jobs\_offered which is an object property of type Job.

#### **5.3.3.4.5 location:**

It has two properties, country which is a data type of type String and country which is also a data type of type String.

#### **5.3.3.4.6 pay:**

It has three attributes, rupee, dollar and pound .These are all of type Integer.

#### **5.3.3.4.7 qualification:**

It has two attributes of type String which are degree and field.

jobs.owl will be imported in your\_ontology.owl for using the instances of classes at third level of hierarchy as job titles. Every job offering company will have to make its own copy of your\_ontology.owl having data related to the jobs offered. While jobs.owl will used as it is by the every company.



## **5.4 Conclusion:**

This chapter incorporated the details of the classes implemented. . The classes have been distributed among the three basic components of the system which are User's machine, Directory and Website Hosting machine. JAVA has been used as the programming language for the project due to its object-oriented and platform independent nature. Also the APIs employed in the project i.e. JADE and Pellet are implemented in JAVA making the system components interoperable

### **6 Testing**

#### **6.1 Introduction:**

Testing is a very important phase in the software development process. Once the coding process is completed, then the software goes under the testing process which involves checking the codes for errors and bugs. It involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results [16]. This chapter involves all the testing techniques which have been employed in the project and the conclusions which have been deduced on the basis of the results of the testing procedures. Test cases for different units and components have been drafted illustrating their expected behaviors on the success and failure of each test. The output of each test is then compared with the one documented in the test case to make sure that the system behaves in the same way in which it is meant to behave.

#### **6.2 Testing Process:**

The testing process has been carried out throughout the development process as an iterative approach has been used in the project for development. Each phase of development was visited several times making sure that the testing process goes in parallel with the development process. The testing was basically done at three levels, Unit testing, Integration testing and System testing.

### 6.2.1 Unit Testing:

Unit testing has been done to determine that whether the individual units of the source program work in the same way in which they are expected to work. The units in the project include those methods which cannot be tested by simple inspection and those classes which cannot be broken down into smaller units for testing. The identified units of the project along with the corresponding test cases are illustrated under the following headings.

#### 6.2.1.1 Launch.java:

It is the class which lies on the user's machine is used to launch the MobileAgent. The expected results on success and failure can be observed from the Table 6.1. On success, the MobileAgent will start and join the main container. On failure, the MobileAgent will not be able to initiate. The tests were successful as it effectively launched the MobileAgent.

**Table 6-1: Test case for Launch.java**

Identity		Launch.java
Category		Unit testing
Description		This class is used to initiate the Mobile Agent.
Set up		JADE set up is needed as a supporting environment.
Expected Results	Success	Mobile Agent starts in a container which then joins the main container.
	Failure	Mobile Agent does not start.

### 6.2.1.2 MobileAgentGui.java:

It is one of the classes that is required on the user's machine. As illustrated in Table 6.2, on success, a GUI appears through which the job preferences of the user are provided to the Mobile Agent. On failure, the GUI is not able to give the window's appearance and feel. When the test was conducted to check this class, the results were quite successful as the same GUI appeared as expected and it successfully took the user's job preferences applying the necessary checks on all the fields.

**Table 6-2: Test case for MobileAgentGui.java**

Identity	MobileAgentGui.java	
Category	Unit testing	
Description	This class takes the user's job preferences through a GUI and gives it to MobileAgent.	
Set up	Dependable classes, MobileAgent and supporting JADE and JAVA swing classes must be present.	
Expected Results	Success	Takes user's job preferences through a GUI, applying checks to all the fields in the GUI such as asking user to enter the salary in numbers if he enters in string.
	Failure	The GUI is unable to give windows look.

### 6.2.1.3 AfterClone () of MobileAgent.java:

MobileAgent.java lies on user's machine. It has many important methods which themselves can be taken as individual units. AfterClone() is one of such methods which dictates the behavior of the MobileAgent clone. If the method executes properly, the MobileAgent clone based on the results returned after querying the ontology, performs

two types of actions. If the result returned is true i.e. the user's data matches with the job offers and requirements in the ontology, the clone writes CV on the website hosting machine and returns the IP to its parent Mobile Agent in the form of an ACL message. Otherwise the clone checks whether it's the last clone and no other clone has sent true, it sends a message to the parent Mobile Agent that no result was found. The test case for this method is shown in Table 6.3. On conducting the tests to check the functionality of this function in MobileAgent.java, results were accurate as the clone applied the right query on the ontology and returned results that were expected.

**Table 6-3: Test case for AfterClone ()**

Identity		AfterClone () of MobileAgent.java
Category		Unit testing
Description		This method runs the Query Engine and dictates the behavior of the MobileAgent clone.
Expected Results	Success	If True is returned, the clone writes CV on the website hosting machine and returns the IP to its parent Mobile Agent in the form of an ACL message. If False is returned, the clone checks whether it's the last clone and no other clone has sent true, it sends a message to the parent Mobile Agent that no result was found.
	Failure	An exception is thrown if there is an error in the query format which cannot occur in our case as the query is being created using the criteria provided by the user in the GUI.

#### **6.2.1.4 onGuiEvent () of MobileAgent.java:**

This method is executed when the user presses the "Search" button on the PersonalDataGui. The test case for this method is illustrated in Table 6.4. On success, this

method provides the MobileAgent with the data taken from the user. It looks up the directory to find the registered websites having job offers in the domain and sub domain as desired by the user. The MobileAgent retrieves the locations of the selected websites from the directory and then clones to the relevant websites. If the method fails to execute properly, the appropriate exception is thrown. Testing this method caused the correct look up of the directory and successful cloning to the selected websites.

**Table 6-4: Test case for onGuiEvent ()**

Identity		onGuiEvent () of MobileAgent.java
Category		Unit testing
Description		It takes the job criteria and personal data from the user, looks up the directory on the basis of domain and sub domain and then sends the clone to the selected website hosting machine.
Expected Results	Success	As the user presses Search button, directory is looked up on the basis of domain and sub domain and clone is sent to the website hosting machine if websites are registered offering jobs in the same domain and sub domain as desired by the user.
	Failure	An exception is thrown if connection is not being able to establish with the directory or if the container running on the website hosting machine closes.

### **6.2.1.5 ReceiveMsg of MobileAgent.java:**

This class is contained inside the MobileAgent.java and receives the ACL message sent by the clone to display the appropriate message or web page to the user. As given in Table 6.5, on its successful execution, it either displays the user a dialog box with message “Sorry no results are found” received from the MobileAgent clone or opens the

webpage for the user from the IP sent by the clone. Tests performed on this class resulted in the correct reception and display of the messages.

**Table 6-5: Test case for ReceiveMsg**

Identity		ReceiveMsg of MobileAgent.java
Category		Unit testing
Description		This class is used to receive the ACL message sent by the clone and displaying the appropriate message or web page to the user.
Expected Results	Success	If no result is found, it displays a message to the user in the form of a dialogue box and disposes of the PersonalDataGui. If result is found, it opens up the web page for the user.
	Failure	An exception is thrown if internet explorer is not able to open the web page.

#### **6.2.1.6 PersonalDataGui.java:**

This unit is also needed on the user's machine. It provides a GUI to the user for getting the user's personal data. The test case for this class is given in Table 6.6. If it works in the right way, then its expected behavior is to takes the user's personal data through a GUI, applying all the necessary checks on GUI's fields. If it is not properly executed, the correct GUI will not be able to appear. The test case in Table 6.6 was used to conduct tests on this unit of the system and the results were very successful as expected results were obtained.

**Table 6-6: Test case for PersonalDataGui.java**

Identity		PersonalDataGui
Category		Unit testing
Description		This class takes the user's personal data through a GUI and gives it to MobileAgent.
Set up		Dependable classes, MobileAgent and supporting JADE and JAVA swing classes must be present.
Expected Results	Success	Takes user's personal data through a GUI, applying checks to all the fields in the GUI such as asking user to fill the first name and last name fields as a compulsion and enter his age and experience in numbers if he enters in string.
	Failure	The GUI is unable to load the windows look and feel.

### **6.2.1.7 PersonalData.java:**

This class is also required to be present on the user's machine. Its test case is illustrated in Table 6.7. On success, its expected behavior is the successful assignment of the user's personal data entered in the PersonalDataGui to its data members through its setter functions and the correct retrieval of the data through its getter functions. The tests conducted on this unit were according to the test case in Table 6.7.

**Table 6-7: Test case for PersonalData.java**

Identity		PersonalData
Category		Unit testing



Description		The user's personal data taken through PersonalDataGui is assigned to its data members and these data members can be manipulated using PersonalData's getters and setters.
Set up		Dependable JAVA classes must be present.
Expected Results	Success	The data entered by the user is successfully assigned to PersonalData's data members through setters and effectively retrieved using its getters when needed.

### 6.2.1.8 Criteria.java:

This presence of this unit is mandatory on user's machine. Its test case is illustrated in Table 6.8. On success, its expected behavior is the successful assignment of the user's job preferences entered in the MobileAgentGui to its data members through its setter functions and the correct retrieval of the data through its getter functions. The results of the test conducted on this class were the same as the expected behavior mentioned in Table 6.8.

**Table 6-8: Test case for Criteria.java**

Identity		Criteria
Category		Unit testing
Description		The user's job preferences taken through MobileAgentGui are assigned to its data members and these data members can be manipulated using Criteria's getters and setters.
Set up		Dependable JAVA classes must be present.
Expected Results	Success	The data entered by the user is successfully assigned to Criteria's data members through setters and effectively retrieved using its getters when needed.

## 6.2.2 Component Testing:

Different units together form a component. After unit testing, the components have been tested to make sure that they behave in the expected way. The test cases for different components of the system are elucidated and shown under the following headings.

### 6.2.2.1 MobileAgent.java:

The presence of MobileAgent.java is obligatory on the user's machine. The test case drafted for it is given in Table 6.9. It takes data from the user, looks up the directory on the basis of job domain and sub domain and clones to the relevant websites' machines. It then queries the ontology. If user's criteria and the ontology match, CV is dropped on that machine and webpage is opened for the user. If they don't match, a message displaying "Sorry, no results are found" is displayed to the user. On success, the expected behavior of this class is the correct execution of all these functionalities. On failure, exceptions are thrown. The tests conducted on this component were very successful as it behaved in the same way as expected according to the test case given in Table 6.9.

**Table 6-9: Test case for MobileAgent.java**

Identity	MobileAgent.java
Category	Component testing
Description	This class takes data from the user, looks up the directory on the basis of job domain and sub domain and clones to the relevant websites' machines. It then queries the ontology. If user's criteria and the ontology match, CV is dropped on that machine and webpage is opened for the user. If they don't match, a message displaying "Sorry, no results are

		found” is displayed to the user.
Set up		Dependable classes, MobileAgentGui, PersonalData, Criteria, QueryEngine and supporting JADE classes must be present. Supporting JADE classes must be added to the class path. Main container of JADE must be running.
Expected Results	Success	Takes user’s job preferences and personal data, looks up the directory, clones to the relevant websites and queries the ontology present there. If user’s criteria and the ontology match, CV is dropped on the website hosting machine and webpage is opened for the user. If user’s criteria and the ontology do not match, a message “Sorry, no results are found” is shown to the user.
	Failure	Exceptions are thrown.

### 6.2.2.2 RegisterAgent.java:

This component is required on the website hosting machine and is needed to register the website’s index page name, ontology’s name, its location, job domain and sub domain with the directory. On success, all this information successfully gets registered with the directory and on failure, the appropriate exception is thrown. The test case for this component is illustrated in Table 6.10. The tests that were performed on this component according to the test case in Table 6.10 were quite successful, validating the test case.

**Table 6-10: Test case for RegisterAgent.java**

Identity	RegisterAgent.java
----------	--------------------

Category		Component testing
Description		This class registers the website's index page name, ontology's name, its location, job domain and sub domain with the directory.
Set up		Supporting JADE classes must be added to the class path. Main container of JADE must be running.
Expected Results	Success	The Register agent finds its container ID (location), gets the website index page's and ontology's name, job's domain and sub domain from the RegisterAgentGui and registers this information with the directory.
	Failure	An exception is thrown if this agent is not able to register with the directory.

### 6.2.2.3 QueryEngine.java:

The existence of this component is compulsory on the user's machine. This class is very significant as it is used by the MobileAgent clone to query the ontology. Its successful execution involves the construction of query on the basis of the data provided by Mobile Agent, applying the query to ontology and returning true if the user's data match with the ontology data and false if it does not. If it does not execute correctly, an exception will be thrown. The test case for this component is illustrated in Table 6.11.

**Table 6-11: Test case for QueryEngine.java**

Identity	QueryEngine.java
Category	Component testing
Description	It queries the ontology on the basis of the data provided by the Mobile Agent and returns true and false depending on whether the user's data match with the data in ontology or not.

Set up		Web server must be running. Supporting Pellet classes must be added to the classpath.
Expected Results	Success	It constructs a query on the basis of the data provided by Mobile Agent. It then applies the query to ontology and returns true if the user's data match with the ontology data and returns false if it does not.
	Failure	An exception is thrown if there is an error in the query format which cannot occur in our case as the query is being created using the criteria given by the user in the GUI.

### 6.2.3 Integration Testing:

Integration testing means testing the functionality of the system stepwise while integrating the components or modules. While amalgamating the components, tests are carried out each time the components are integrated. If the tests are successful, then further integration of the system takes place. Otherwise the components are debugged and integrated again and again until the tests are successful.

In the project, the components were integrated in three main steps. First of all Directory and the Register Agent were integrated and testing was done. If the test results were successful, then Mobile Agent was combined with these two and the system was tested again. After that, Query Engine was amalgamated with the rest of the system and tests were carried out again. These steps have been elaborated as follows:

#### 6.2.3.1 Integration of Directory and RegisterAgent:

RegisterAgent here means all the classes that are needed for registration of the companies with the directory i.e. Launch.java, RegisterAgent.java and RegisterAgentGui.java. Initially the RegisterAgent and Directory were integrated and their combined functionality was tested. Through RegisterAgent, the job companies

registered their job domain, sub domain, location, webpage name and ontology name with the directory facilitator agent of the main container. Then we checked the results that whether the companies got registered with the directory or not. The results were correct, so further components were integrated with it. The test case which was considered to check the expected behavior of the integrated components.

**Table 6-12: Test case for Integrated RegisterAgent and Directory**

Identity		Integrated RegisterAgent.java and Directory
Category		Integration testing
Description		As these classes are integrated, the combined functionality they perform is the registration of company's website's index page name, ontology's name, its location, job domain and sub domain with the directory.
Set up		Supporting JADE classes must be added to the class path. Main container of JADE must be running.
Expected Results	Success	The Register agent finds its container ID (location), gets the website index page's and ontology's name, job's domain and sub domain from the RegisterAgentGui and get registered successfully with the Directory Facilitator Agent of the main container.
	Failure	The job companies are not able to get registered with the directory (DF Agent).

### 6.2.3.2 Integration of Directory, RegisterAgent and MobileAgent:

MobileAgent here means collection of those classes which are employed which are used to take the user's data, look up the directory, clone to selected websites, query the ontology, drop the CV and return the results to the user. The set of classes included are Launch.java, MobileAgent.java, MobileAgentGui.java, Criteria.java, PersonalData.java and PersonalDataGui.java. During integration testing, the MobileAgent was integrated

with the Directory and RegisterAgent and their collective functionality was tested. The results were quite successful. The MobileAgent obtained the user's job preferences and personal data and successfully looked up the directory to get the results and cloned to the relevant websites hosting machines. The tests were conducted taking into account the following test case.

**Table 6-13: Test case for Integrated MobileAgent, RegisterAgent and Directory.**

Identity		Integrated MobileAgent, RegisterAgent and Directory
Category		Integration testing
Description		As these classes are integrated, the collective functionality which is performed includes the Mobile Agent taking data from the user, looking up the directory on the basis on the basis of domain and sub domain and then cloning to the relevant websites' hosting machines. Also the working requires the successful registration of websites with the directory.
Set up		Dependable classes, MobileAgentGui, PersonalData, Criteria and supporting JADE classes must be present. Supporting JADE classes must be added to the class path. Main container of JADE must be running.
Expected Results	Success	The MobileAgent is able to take the user's data, look up the directory on the basis of domain and sub domain, retrieve the location of the website hosting machine and clone to those machines.
	Failure	The MobileAgent is not able to take the user's data or look up the directory on the basis of domain and sub domain or it might not be able to clone to the retrieved locations of the website hosting machines.

### 6.2.3.3 Integration of Directory, RegisterAgent, MobileAgent and

#### QueryEngine:

This is the final stage of integration when all the basic components are integrated together and checked for their final working. The results were very encouraging. The clones were enable to query the ontology using QueryEngine and return the predicted results to the user. The test case which was written to authenticate the combined functionality of the system is as follows:

**Table 6-14: Test case for Integrated MobileAgent, RegisterAgent, Directory and QueryEngine**

Identity		Integrated MobileAgent, RegisterAgent, Directory and QueryEngine
Category		Integration testing
Description		The combined functionality includes the MobileAgent taking data from the user, looking up the directory on the basis of job domain and sub domain and cloning to the relevant websites' machines. It then queries the ontology. If user's criteria and the ontology match, CV is dropped on that machine and webpage is opened for the user. If they don't match, a message displaying "Sorry, no results are found" is displayed to the user. The working also includes the registration of websites with the directory.
Set up		Dependable classes, MobileAgentGui, PersonalData, Criteria, QueryEngine, RegisterAgentGui and supporting JADE classes must be present. Supporting JADE classes must be added to the class path. Main container of JADE must be running.
Expected Results	Success	The MobileAgent clone queries the ontology present on website hosting machines. If user's criteria and the ontology match, CV is dropped on the website hosting machine and webpage is opened for the user. If user's criteria and the ontology do not match, a message "Sorry, no results are found" is shown to the user.



	Failure	The MobileAgent clone is not able to query the ontology present on website hosting machines.

#### **6.2.4 White Box Testing:**

White box testing or structural testing uses an internal perspective of the system to design test cases based on internal structure. It requires programming skills to identify all paths through the software [17].The white box testing of the system has been done at both unit testing and component testing stages.

#### **6.2.5 Black Box Testing:**

Black Box Testing is testing without knowledge of the internal workings of the item being tested [18]. It attempts to derive sets of inputs that will fully exercise all the functional requirements of a system. For each set of inputs, outputs are known and in black box testing, the inputs are fed in and if the output matches the predicted output it means that the system delivers the expected functionality.

If we consider that data as valid data for which the job offers are available and the data for which the companies are not offering jobs as invalid data, following tests were conducted as part of the black box testing.

##### **6.2.5.1 Checking the System on Valid Data:**

First the system was checked on data for which the registered companies were offering jobs. The webpage of the website offering that job was returned which proved the right functionality of the system on entering valid data.

### **6.2.5.2 Checking the System on Invalid Data:**

Secondly that set of data was entered for which the job offers were not available. In this case the Mobile Agent displayed the message “Sorry, no results are found” showing that the system worked correctly on entering the invalid data.

#### **6.2.5.2.1 Skipping the noncompulsory fields:**

The query on ontology is done even if all the fields are not filled except domain, sub domain, first name and last name as these are compulsory fields. The null fields are ignored and the query which builds at run time on the back end does not include these fields in the query. To check that whether the system was delivering this functionality, some of the fields, except the compulsory fields, were skipped. The query was done and it was observed that correct results were returned by the Mobile Agent, authenticating the right functionality of the system on skipping the noncompulsory fields.

#### **6.2.5.2.2 Skipping the compulsory fields:**

Next while entering data, compulsory fields were skipped and it was observed that, messages were displayed prompting to fill the compulsory null fields, verifying that the system was correctly implementing this functionality as well.

### **6.2.6 Static Analysis of Code:**

Besides testing the code dynamically, static analysis of the code has been done as well to find defects, if any, in the blocks of code due to which it does not implement the exact requirement or to determine the ways by which the code can be optimized to make it fool proof.

The code has been statically analyzed in many ways which are briefly illustrated under following headings.

### **6.2.6.1 Control Flow Analysis**

Control flow analysis has been carried out for the verification and validation of control blocks in the source code, for instance, the ‘for’, ‘while’ loops and the ‘if’ condition blocks. It has been observed that no unnecessary code has been included and all these blocks are optimized.

### **6.2.6.2 Data Analysis:**

Data analysis has been done to find and remove improper initializations, unnecessary assignments and those variables that are declared but never used. All such unnecessary lines have been eliminated thus giving a refined code.

### **6.2.6.3 Interface Analysis:**

Interface analysis has also been done to ensure consistency of interface, class, procedure declaration, definition and their use. It has been observed through tests that all the methods declared in the interface are correctly implemented in the classes and that there are no redundant methods.

### **6.2.7 Conclusion:**

This chapter illustrated the testing process of the system that has been carried out and the corresponding results obtained. The testing of the system has been done in great detail. The test cases have been written for the three main phases of testing, unit testing, component testing and integration testing. Using these test cases, the results of the tests

have been authenticated. Both white box and black box testing have been carried out to determine that whether the system delivers all the functional requirements that it should be delivering. Even static inspection of the code has been carried out as well so that it become optimized and does not become redundant. All the test results were very successful proving that the system delivers all its functionalities in an efficient way.

### **7 Future Work and Conclusion**

#### **7.1 Future Work:**

The system that has been implemented in the project can be extended in many ways. First of all, the search which now goes on is in terms of job seeking. In future the search can be made generalized as is done in any general purpose search engine. Also based on the design of the proposed system, the search facility can be extended for other application domains as well.

When this project was started, JADE supported only intra-platform mobility. So the system has been designed in such a way that all agents register with one main container, making all of them part of the same main container. But now the JADE supports inter-platform mobility as well. So in future the project can be extended by running more than one main container which means more than one directory will be there enabling the MobileAgent to look up and utilize the services of the agents of other platforms as well.

#### **7.2 Conclusion:**

Semantic Web and Mobile Agents are two promising technologies that have been progressing at an enormous pace. The system has been developed for automated searching and filtering of information on the web for the purpose of job seeking - based on these two technologies. It aims to provide precise results as well as convenience to the user; making the application very significant as there is a drastic need of such automated solutions in the fast and demanding world of today.

## **APPENDIX A**

### **Research Paper**

# Personalized Mobile Agent Based Searching of Semantic Web

Sara Rehmat, Amina Khan, Fatima Naeem, Urooj Saeed,  
Athar Mohsin, Umar Mahmud,

CS Department, Military College of Signals, National University of Sciences and  
Technology, Rawalpindi, Pakistan

## Abstract

This paper illustrates the integration of Mobile Agent and Semantic Web technologies by proposing a solution that provides domain specific search to user. The user uses Mobile Agent to search for his required information on Semantic Web and get more meaningful and refined results in return.

## Keywords

Semantic Web, Mobile Agent, Semantic Search Engine

## 1. Introduction

World Wide Web is the universe of information most of which is for humans' perusal, not for computer programs to manipulate meaningfully. Semantic Web is a promising technology that is aimed at putting meanings into the contents of current Web, thus enabling the computer programs to understand the contents and consequently process them intelligently [1]. The main feature of Semantic Web is the existence of metadata that facilitates the meaningful and intelligent search by computer programs.

Mobile Agents are programs that can autonomously travel across a network and perform tasks on machines that provide agent hosting capability [2]. The use of mobile agent reduces network load as instead of downloading large amount of data for processing, the processing is transferred to where data lies.

## 2. Related Work

Semantic Web is evolving at a great pace which generates the need for search engines that can efficiently search for the Semantic Web documents i.e. documents written in RDF or OWL. One of the popular semantic search engines is Swoogle. It is a crawler based indexing and retrieval system for the Semantic Web documents [3]. Swoogle has some limitations like it provides only weak access to semantic content, which is keyword based search that doesn't consider the semantic content of documents it accesses. Swoogle doesn't consider semantic relations between ontology other than the ones that are explicitly stated (e.g. import) [4]. SemSearch is another semantic

search engine that provides to the end users an easy mechanism to formulate complex queries and produces more accurate results [5]. But this system is a closed world system i.e. it doesn't interact with the web and it can not make use of knowledge from other repositories. Sindice is a Semantic Web Crawler that indexes and crawls Semantic Web documents in order to find the places where particular URI is mentioned. In Sindice only URIs and not keywords can be entered into the search box [6]. All these search engines are crawler based search engines. With the growing size of Semantic Web, the crawlers have to continuously retrieve a very large number of documents. Contrary to crawlers, the suggested solution uses Mobile Agent to access and process the Semantic Web documents only in response to users' queries which results in comparatively less network bandwidth consumption. Moreover, this system provides domain specific search. In [7], a project is proposed that also searches within job domain but that approach uses static intelligent agents.

### 3. Scenario

Ali is searching for a job that matches his preferences like salary, timings, city or country etc. He searches on WWW using a conventional search engine. The search engine will return a large number of results where the words he entered are present in any combination and many websites that don't contain the exact words as entered by him will not be shown as part of the search result though they have job descriptions matching user's preferences. Consequently many of the results

returned by the search engine will be irrelevant and of no use to Ali.

### 4. Proposed Solution

The solution we propose uses two technologies – Mobile Agent and Semantic Web. Instead of user searching on World Wide Web himself, he provides his job preferences to a mobile software agent that searches on user's behalf. The mobile agent moves to the machines hosting websites of companies providing jobs. These machines must have a supporting environment to allow mobile agents to come.

The mobile agent first looks up a directory on the basis of the job domain entered by user like IT, medicine, management etc. The mobile agent will get a list of relevant websites which it starts visiting one by one.

To enable the mobile agent to do the meaningful search, semantics will be put in the websites through the use of ontology. The mobile agent will process the ontology thus the resulting search will not be on the basis of matching words but on the basis of matching concepts in user's preferences and job ontology. Consequently, the results returned by the mobile agent will provide more useful information to the user achieving the true spirit of World Wide Web.

The design of the proposed solution is shown in Fig 1.



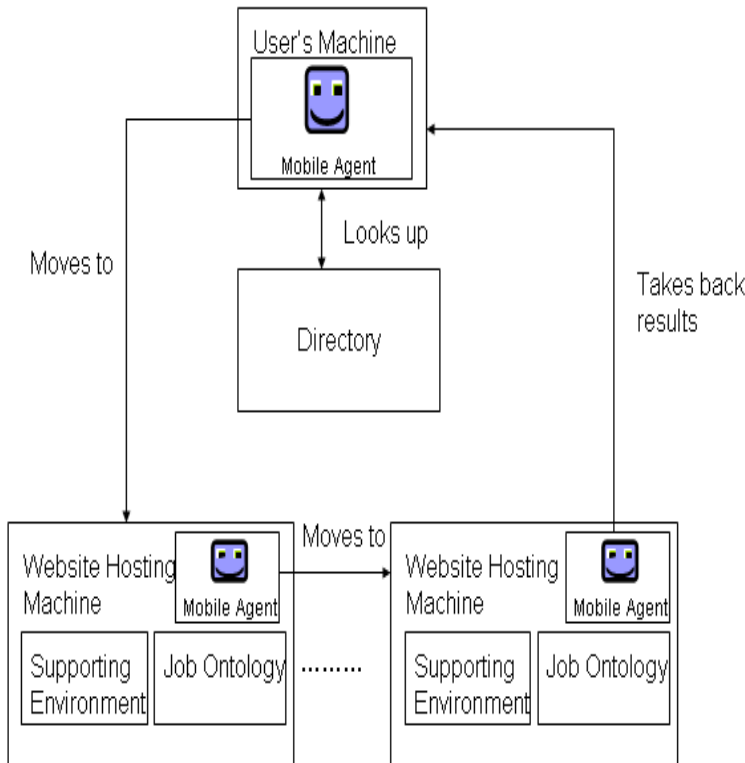


Figure 1: Proposed Design

## 5. Conclusion

The proposed solution uses Mobile Agent to search for user's job preferences on Semantic Web. The results returned by the Mobile Agent are more precise as searching on Semantic Web is done on the basis of matching concepts contrary to matching strings in the conventional search engines.

## References

[1] By Tim Berners-Lee, James Hendler and Ora Lassila, "The Semantic Web," <http://www.sciam.com/article.cfm?id=the-semantic-web>, 26 November, 2007.

[2] Ashish Malgi, Neelesh Bansod and Byung Kyu Choi, "STRING: Efficient Implementation of Strongly Migrating Mobile Agents in Java," <http://www.cs.mtu.edu/~npbansod/58.pdf>, November 10, 2007.

[3] Tim Finin, et al., "Swoogle: A Search and Metadata Engine for the Semantic Web," <http://www.cs.umbc.edu/~ypeng/Publications/2004/swoogle.pdf>, November 15, 2007.

[4] Laurian Gridinoc, Mathieu d'Aquin, Davide Guidi, Martin Dzbor and Enrico Motta, "PowerMagpie: A Semantic Web Browser – v1 OpenKnowledge Deliverable D8.1," [http://powermagpie.open.ac.uk/ok-d8.1/OK\\_D8\\_1\\_PowerMagpie.pdf](http://powermagpie.open.ac.uk/ok-d8.1/OK_D8_1_PowerMagpie.pdf), September 25, 2007.

[5] Yuanguai Lei, Victoria Uren and Enrico Motta, "SemSearch: A Search Engine for the Semantic Web," [http://kmi.open.ac.uk/publications/pdf/semsearch\\_paper.pdf](http://kmi.open.ac.uk/publications/pdf/semsearch_paper.pdf), October 2, 2007.

[6] Afraz Jaffri, "Searching on the Open Semantic Web Using a URI Identity Management Approach," <http://eprints.ecs.soton.ac.uk/14428/1/Mi ni-Thesis.pdf>, October 20, 2007.

[7] Zuzana Halanová, Pavol Návrát and Viera Rozinajová, "A Tool for Searching the Semantic Web for Supplies Matching Demands," <http://ecet.ecs.ru.acad.bg/cst06/Docs/cp/SII/II.16.pdf>, November 1, 2007.

## **APPENDIX B**

### **Hardware and Software Requirements**

# Hardware and Software Requirements

## Hardware Requirements:

- 1.0 GHz Processor or More
- 256 MB of RAM or More

## Software Requirements:

- **Platform:**

jdk-1.5.0

- **Operating System:**

Windows ® XP/98/2000/NT 4.0

## **APPENDIX C**

### **User Manual**

# User Manual

This application software has two categories of users.

1. A person who is looking for a job and uses this software to get precise and more refined results as compared to those returned by conventional search engines.
2. Companies and organizations that are offering jobs and want their job descriptions to be semantically searched by users.

## **1. For Job Seekers:**

To start the application, run **start.bat** batch file in **Mobile Agent** folder as described in deployment manual.

### **Setting Job Criteria:**

For the following steps refer to Fig 1.

1. Select the domain of job you are looking for. It is mandatory to select one of the fourteen domains. Otherwise an error message will be displayed asking you to select a domain in order to proceed further.
2. For any domain selected in step 1, select a sub domain in step 2. If you don't make any selection, the first sub domain displayed for any domain will be selected as your choice.
3. Enter the title of your desired job. Examples of titles can be programmer, project manager, test engineer etc. You can leave this text field blank.
4. Enter your choice of company or institution. If you don't want to specify the organization, you can leave this field blank.
5. Specify whether you are looking for a full time job or a part time one. If you don't have any reservations regarding the timings of job, select 'any' as your choice.

The screenshot shows a web browser window titled "Job Hunt" with a purple header that says "DREAM JOB SEARCH". Below the header, there is a pink background with a pattern of overlapping circles. The text "Please enter the criteria for the job you are looking for." is displayed. The form contains the following elements:

- Job Domain:** A dropdown menu with "--Select--" selected. A line labeled "1" points to this dropdown.
- Sub-Domain:** A dropdown menu. A line labeled "2" points to this dropdown.
- Title:** A text input field. A line labeled "3" points to this field.
- Company/Institution:** A text input field. A line labeled "4" points to this field.
- Timings:** Three radio buttons labeled "Part Time", "Full Time", and "Any". The "Full Time" radio button is selected. A line labeled "5" points to this group.
- Minimum Pay:** A text input field. A line labeled "6" points to this field.
- Currency:** A dropdown menu with "Rs" selected. A line labeled "7" points to this dropdown.
- City:** A text input field. A line labeled "8" points to this field.
- Country:** A text input field. A line labeled "9" points to this field.
- Next:** A red button with the text "Next". A line labeled "10" points to this button.

**Fig 1**

6. Provide the minimum amount of expected salary in numbers. As with other text fields, you can leave this text field blank. In case you enter non numeric characters, you'll be asked to enter the salary in numbers only.
7. Select the currency of salary you have entered in step 6. You don't have to select the currency in case you haven't provided the salary.
8. Give the city name where you want the job.
9. Give the country name where you want the job.

10. Click **next** button to proceed further.

## **Setting Personal Data:**

For the following steps refer to Fig 2.

1. Provide your first name and last name respectively. These text fields are compulsory to fill as indicated by the asterisks.
2. Specify your age in numbers. This field is not compulsory to fill. If you enter non numeric characters, you'll be asked to enter age in numbers.
3. Select the highest degree you have earned. Select **–Select–** from the drop down list if you don't want to specify your degree.
4. Provide the field in which you have earned your degree. This field can be left blank.
5. Mention your experience in years in numbers otherwise you can't proceed further if you enter non numeric characters. If you don't have experience or don't want to mention it, this field can be left blank.
6. Give your contact information that can be either your phone number, postal address or email address etc. This field is not necessary to fill.
7. If you want to have your Curriculum Vitae dropped at machines hosting websites of companies providing jobs of user's choice, paste your CV into the text area otherwise you can leave it blank.
8. Click **Start** to start your search for your dream job.

The image shows a software window titled "Personal Data" with a red header bar containing the text "DREAM JOB SEARCH". The main area has a yellow grid background and contains several form fields:

- 1: Two text input fields labeled "First Name" and "Last Name".
- 2: A text input field labeled "Age" followed by the text "years".
- 3: A dropdown menu labeled "Qualification Degree" with "--Select--" and a downward arrow.
- 4: A text input field labeled "Field".
- 5: A text input field labeled "Experience" followed by the text "years".
- 6: A text input field labeled "Contact Info".
- 7: A large, empty rectangular text area labeled "Paste CV".
- 8: A rounded rectangular button labeled "Search".

**Fig 2**



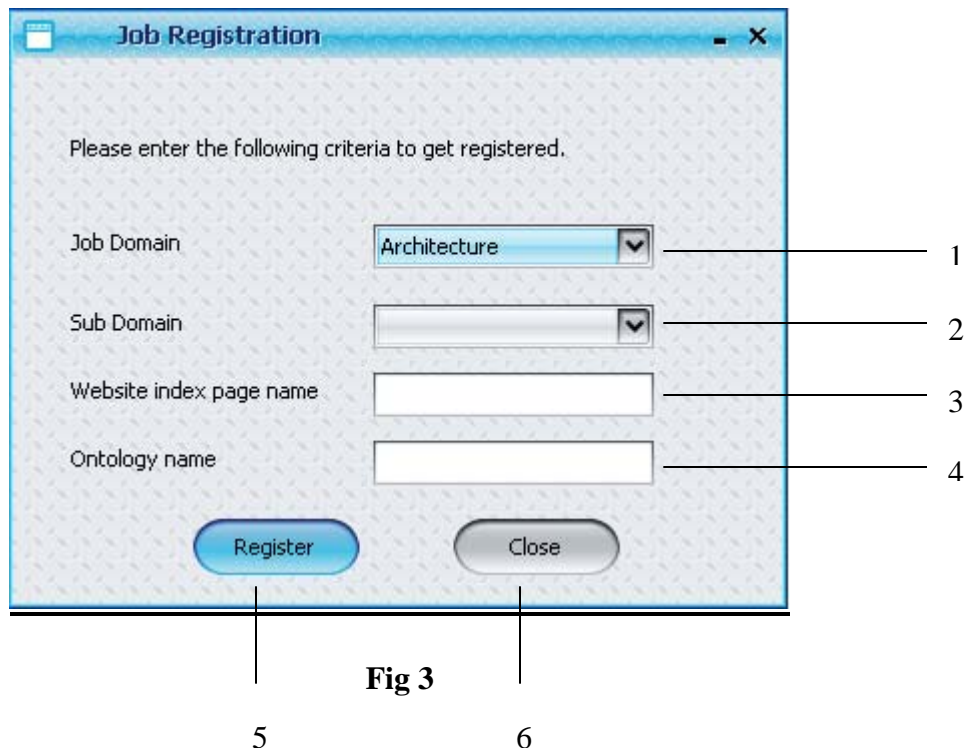
## 2. For Job Employers:

To start the application run **start.bat** in **Register Agent** folder as described in deployment manual.

### Registering Jobs:

For the following steps refer to Fig 3.

1. Select the domain of job you are registering.
2. For any domain selected in 1, select a sub domain in 2. If you don't make any selection, the first sub domain displayed for any domain will be selected as your choice.
3. Enter the name of website in which the jobs your company is offering are advertised.
4. Enter the name of ontology your company is using.
5. Press **Register** button to get your job registered for semantic search by this application.
6. After registration click **Close** button to close this GUI.



**APPENDIX D**  
**Deployment Manual**

# Deployment Manual

This software has different components that need to be installed on different computers for the application to run.

## **Deployment on User's Machine:**

User who is looking for a job and wants to use this application to get meaningful and more refined results as compared to those returned by conventional search engines needs to follow the following steps.

1. Unzip the folder **Job Search.zip** to extract the folders **jade**, **pellet-1.5.1**, **Mobile Agent** and batch file **start.bat**.
2. Copy **jade** and **pellet-1.5.1** folders on your computer.
3. Open **start.bat** batch file for editing. Replace <path> by the path of the folder where you have copied **jade** and **pellet-1.5.1**. Replace <java-home> by the path of folder where folder **jdk-1.5.0** is placed. Save it.
4. Copy the batch file **start.bat** in **Mobile Agent** folder.
5. Run this batch file to start the application.

## **Deployment on Job Server:**

This application requires JADE Main Container to be running on some computer which is referred as Job Server in this manual. Following steps need to be taken to run JADE Main Container.

1. Unzip the folder **Directory.zip** to extract folder **jade** and batch file **launch.bat**.
2. Copy **jade** folder on your computer.
3. Open **launch.bat** for editing. . Replace <path> by the path of the folder where you have copied **jade**. Replace <java-home> by the path of folder **jdk-1.5.0** is placed. Save it.

4. Launch Main Container by running the batch file **launch.bat**.

### **Deployment on Website Host:**

The companies or organizations that are offering jobs and want to have their job descriptions semantically searched by users have to take the following steps. The machine where they are hosting their websites is referred to as website host. **Protégé 3.2.1** or higher and **jswdk** server must be installed on this machine.

1. Unzip the folder **Register.zip** to extract the folders **jade**, **pellet-1.5.1** and **Register Agent**, batch file **start.bat** and two owl files **jobs.owl** and **your\_ontology.owl**.
2. Open **your\_ontology.owl** with **Protégé-3.4** and make an instance for each job your company is offering. Set the values of properties of each job instance. Refer to **Protégé** guide if you are not familiar with using **Protégé**. Save the file.
3. Place both the owl files in the webpages directory of jswdk server.
4. Run jswdk server.
5. Copy **jade** and **pellet-1.5.1** folders on your computer.
6. Open **start.bat** batch file for editing. Replace <path> by the path of the folder where you have copied **jade** and **pellet-1.5.1**. Replace <java-home> by the path of folder where **jdk-1.5.0** is placed. Save it.
7. Copy the batch file **start.bat** in **Register Agent** folder.
8. Run this batch file every time you want to register a job.

## **APPENDIX E**

### **Symbols And Abbreviations**

## **Symbols And Abbreviations**

**WWW:** World Wide Web

**JADE:** Java Agent Development Environment

**AMS:** Agent Management System

**DF:** Directory Facilitator

**ACL:** Agent Communication Language

**XML:** Extensible Mark up Language

**XMLS:** Extensible Mark up Language Schema

**HTML:** Hyper Text Mark up Language

**DTD:** Document Type Definition

**W3C:** World Wide Web Consortium

**RDF:** Resource Description Framework

**RDFS:** Resource Description Framework Schema

**OWL:** Web Ontology Language

**GUI:** Graphical User Interface

**CV:** Curriculum Vitae

**URL:** Uniform Resource Locator

**FIPA:** Foundation for Intelligent Physical Agents

**UML Diagrams:** Various diagrams including Use-Case, Sequence and Data Flow that show the initially perceived outline functionality of the intended system.

## **APPENDIX F**

### **Bibliography**

## References

- [1] “Introduction-The Semantic Web,”  
[http://www.ryerson.ca/~dgrimsha/courses/cps720\\_02/intro.html](http://www.ryerson.ca/~dgrimsha/courses/cps720_02/intro.html)
  
- [2] Franklin and Art Graesser, “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents” in Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, Springer-Verlag, 1996.
  
- [3] “Mobile Agent,” [http://en.wikipedia.org/wiki/Mobile\\_agent](http://en.wikipedia.org/wiki/Mobile_agent)
  
- [4] “Seven Good Reasons for Mobile Agents,” <http://www.moe-lange.com/danny/docs/7reasons.pdf>
  
- [5] Giovanni Caire, “JADE Programming For Beginners,” <http://jade.tilab.com/doc/JADEProgramming-Tutorial-for-beginners.pdf>
  
- [6] Tim Berners-Lee, James Hendler and Ora Lassila, “The Semantic Web,” in Scientific American: Feature Article: The Semantic Web: May 2001.
  
- [7] “XML Primer,” [http://www.w3schools.com/web/web\\_xml.asp](http://www.w3schools.com/web/web_xml.asp)
  
- [8] “Introduction to RDF,” [http://www.w3schools.com/rdf/rdf\\_intro.asp](http://www.w3schools.com/rdf/rdf_intro.asp)
  
- [9] “Ontology (Information Science)”  
[http://en.wikipedia.org/wiki/Ontology\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Ontology_(computer_science))
  
- [10] “OWL Web Ontology Language,” <http://www.w3.org/TR/owl-features/>
  
- [11] “Protégé,” <http://protege.stanford.edu/overview/>
  
- [12] “Jena Documentation,” <http://jena.sourceforge.net/documentation.html>
  
- [13] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, Yarden Katz, “Pellet-A Practical OWL-DL Reasoner”
  
- [14] “Pellet,” <http://pellet.owldl.com/faq/query-engines/>
  
- [15] “Pellet, FAQ,” <http://pellet.owldl.com/faq/single-page#using-pellet-in-jena>



- [16] Jiantao Pan, "Software Testing," Carnegie Mellon University, 18-849b Dependable Embedded Systems, Spring 1999  
[http://www.ece.cmu.edu/~koopman/des\\_s99/sw\\_testing/#introduction](http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/#introduction)
- [17] "White Box Testing," [http://en.wikipedia.org/wiki/White\\_box\\_testing](http://en.wikipedia.org/wiki/White_box_testing)
- [18] Thomas Raishe, "Black Box Testing," Courses for CEN4010-SE 2002, Computer Science and Engineering Department, Florida Atlantic University.  
<http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C13/black.html>