Intelligent Multi-Agent Online Examination System

Jane Chioma Anaekwe

Submitted to the
Institute of Graduate Studies and Research
in partial fulfilment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
February 2015
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Serhan Çiftcioğlu
Acting Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quantity as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Zeki Bayram
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Zeki Bayram _____
2. Assoc. Prof. Dr. Alexander Chefranov _____
3. Asst. Prof. Dr. Adnan Acan _____

ABSTRACT

Examinations are essential tools in an education system and are used to test the knowledge, learning capabilities, and progress of a student in a specific domain. Due to its importance in an academic system, it is therefore vital to have an examination system that is fair and efficient. The traditional paper-based examinations are known to have various constraints such as, their time consuming nature, delays in declaration of results, human errors and tedious evaluation/management of large amounts of paper which are prone to security risks and exam malpractice. Due to these limitations the paper based systems have received various criticisms about its efficiency and fairness.

This thesis is aimed at solving the problem of the traditional paper-based exams by the introduction of an intelligent multi-agent online examination system. The system is a paperless exam integrated with intelligent agents. Each agent introduced in the system has specific duties that they perform and are capable of communicating with each other thereby forming an interconnected network. The duties of the agents include monitoring exams, providing various services during exams e.g. extra time requests, teacher requests, instant calculation and display of results, performance analysis, speed analysis, and suggestions to students and teachers on topics students need to improve in. Simulation exams (practice exams) are also provided to help students improve their performance and become familiar with the system. This approach provides a fair and efficient way of handling exams with reduced cost, workload, errors and risks thereby providing assistance and convenience to both teachers and students.

Keywords: Multi-agent system, online examination system, efficient services.

ÖZ

Sınavlar, bir öğrencinin belli bir alandaki bilgisini, öğrenme becerilerini, ve ilerlemesini test etmeye yarayan elzem araçlardır. Akademik sistemdeki öneminden dolayı, adil ve verimli bir sınav sistemi önemlidir. Geleneksel kağıt tabanlı sınavlarda çeşitli kısıtlamalar olduğu bilinmektedir. Bunlar arasında zamanın boşa harcanması, sonuçların bildiriminde oluşan gecikmeler, insan hataları, büyük miktarda kağıdın değerlendirilmesi/idaresi'nin sıkıcı ve güvenlik yönünden riskler taşıması sayılabilir. Bu sorunlardan ötürü kağıda dayalı sistemler verimlilik ve adaletlilik yönünden eleştiriler almışlardır.

Bu tezin amacı, akıllı, çok etmenli, çevrimiçi bır sınav sistemi ile geleneksel kağıt tabanlı sınvaların yol açtığı sorunlara çözüm getirmektir. Sistem akıllı etmenlerle entegre olmuş kağıtsız sınavlardan oluşur. Sistemdeki her etmenin kendine has görevleri vardır ve etmenler birbirleri ile konuşabilmeleri sayesinde bağlantılı bir ağ oluştururlar. Etmenlerin görevleri arasında sınavları gözetleme, sınav esnasında bazı hizmetler sunma (örneğin ek süre taleplerini değerlendirme), öğretmen talepleri, sonuçların anında hesaplanıp gösterilmesi, performans analizi, hız analizi, ve öğrenciler ile öğretmenleri öğrencinin gelişim göstermesi gereken konularda bilgilendirmesi vardır. Sistemde var olan deneme sınavları da öğrencilerin perrformansını arttırmaya ve sisteme aşinalık kazanmaya yardımcı olur. Bu yaklaşım sayesinde sınavların adaletli, verimli, azaltılmış maliyet, iş, hata ve risk ile yapılması mümkün hale gelmektedir.

Anahtar kelimeler: Çok etmenli sistem, çevrimiçi sınav sistemi, verimli hizmetler

To God Almighty, the
help of my
countenance

ACKNOWLEDGEMENT

I would like to thank God Almighty for His guidance, support, faithfulness and love throughout my thesis research and course of study. Without Him it would never have been possible. To Him be all the glory and honour.

I would also like to thank my supervisor Assoc. Prof. Dr. Zeki Bayram for his patient support and advice which have been a useful guide during the preparation of my thesis.

My gratitude and thanks also goes to my family and friends who have played a very vital role in all my academic endeavours. Their love, support and care gave me strength to keep going and has been a great source of encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

PBE	Paper-based Examination
OES	Online Examination Systems
PC	Personal Computer
FIPA	Foundation for Intelligent Physical Agents
JADE	Java Agent Development Framework
MAS	Multi-agent Systems

Chapter 1

INTRODUCTION

1.1 Limitations of Paper-based Examinations

The Paper-based examination (PBE) system has been used for centuries in evaluating the learning abilities of a person. The introduction of written exams is believed to have started in China over a thousand years ago [1] and are still in use today. The PBE system, to some extent, enables one to assess how much a person knows and how well he or she is able to apply what they know in solving problems [2]. Although the PBE system has been useful in many ways various shortcomings have been detected which limits its efficiency. PBEs are to prone to human errors and limitations. Mistakes in the grading of an exam and delays in declaration of results can occur which raises suspicions of unfairness [2], [3]. A tedious amount of time goes into co-ordination of paper-based exams, the distribution and collection of large amounts of papers, and the manual assessment of these papers can make the whole process inefficient and monotonous [2], [3], [4]. The rigidity of the exam location and exam time is another setback in the PBE system, times and locations are usually fixed and can't be altered easily [2], [3].

Other limiting factors of PBEs are their cost and security [3], [4], [5]. Printing large amounts of exam papers, hiring extra hands (e.g. invigilators), and safe guarding of

exam papers are issues which plague the PBE system, and create potential problems [3], [4], [5].

Since examinations are still of great importance in our society today, a more efficient way of handling them is needed and as we go on a better alternative is shown.

1.2 Online Examination Systems

With the growing rate of development in Internet and network technologies, many organizations are now migrating from a manual system of operation to a computer based system. Academic organisations are not an exception as many academic institutions are now replacing their PBE systems with online examination systems (OES) [5].

Online examination systems are paperless testing systems through which students or individuals can take tests or exercises via a computer with the use of Internet and network technologies [6], [7], [8]. Examination questions, student answers, exam results and other necessary information are passed across to system users via a network, and the operations involved in managing an examination are handled automatically by the system [6] [8].

With the new developments in online exams, OES is gradually becoming the preferred choice over PBE systems [3], [4], [5]. Online examination systems provide ease and convenience by reducing the workload, cost and limitations of the PBE systems. They reduce the time costs since most functions such as, the distribution of questions, grading, and display of results are handled instantly [9]. Online examinations can be

taken at different times and in different locations which solves the time-space constraints of the paper-based system [2], [3], [4], [8], [9] and opens up opportunity for distance learning.

The advanced features in OES improves the quality of teaching and learning [9] and exposes students to a system that is fast, efficient, and objective [3], [4], [7], [9].

1.3 Agent-based Online Examination System

There are various ways in which online examination systems are implemented and each have their own method of operation. Ahmad, Khan and Abbas presented a PHP and MySQL based online examination system with power failure handling and Dropbox capability [5], Li and Wu Presented an online examination system based on Browser/Server (B/S) structure [8], Liu and Wang presented an online examination system based on UML modelling and MVC design pattern [9], Lu and Hu presented the design and implementation of an online examination system based on J2EE [10], and Li Jun presented the design of an online examination system based on Web service and COM [11]. These articles amongst others describe various OES implementations using different methodologies, each with its own advantages and disadvantages (described in section 1.4). The focus of this thesis will be on the agent-based methodology.

Agent technologies have been applied in a wide variety of areas such as, information retrieval from databases and the internet, information filtering, intelligent user interfaces, decision making, smart messaging, electronic business etc. [12], [13]. One

of the aims of using agent technology is to introduce autonomy and intelligence into applications by making the agents perform tasks on behalf of users [12].

In this thesis, intelligent agents will be used in implementing an online examination system in which agents perform various tasks on the system making it smart and efficient. The system is divided into various modules with agents performing specific duties on them. The agents work hand in hand and communicate with one another in order to accomplish specific goals. Each duty that the agents perform play an important role in the total performance of the system. The implementation of the agent system was done using the JADE agent platform [14]. Further details on the implementation of the system will be discussed later on.

1.4 Related works and Comparisons

Various works on online examination systems have been implemented each with its own methodology and mode of operation.

- Bhuvaneswari, Sujatha and Deepthi published a proposal on centralized Exam Assesment Aportion using Mobile agents [15]. The article deals with implementing an agent-based system for handling centralized exam assessment with the aim of improving information retrieval in distributed systems and enhancing the security of data.

  Comparison: Although the system has a secure mechanism for protecting data (Tamper detection framework) and a well-structured architecture, it has little focus on providing mediums through which students can build up their learning skills and improve their academic performance. This thesis aims at providing these mediums through practise exams, agents suggestions that help students analyse

their performance and useful agent assistance offered to students during exams (eg hurry up reminders, calculating student's estimated time to finish, enabling students make request during exams). More on these features can be seen in chapter 3 and 4.

- Gawali and Meshram [2] designed an agent-based autonomous examination system using Java aglets. The system makes use of main, mobile and stationery intelligent agents located in its modules (Examination, Authentication, result calculation) to perform various tasks. One of the aims of the research was use the idea of agent mobility and agent communication to create a system that increases network performance and reduces network delays.

  Comparison: Although the proposed system reduces certain limitations there is still little mentioned on building a student/teacher-friendly system that meets he needs of both teachers and students. This thesis aims at providing a system that students and teacher can perform various useful activities in. The system provides useful moodles design particularly for teacher and students. These moodles (e.g. setting practise/exam questions, viewing agent performance analysis, viewing/handling student exam requests, taking practise exams) provide a conducive and convenient environment for students and teachers to perform various operations effectively, hence improving their performance.

- Ali and Hussain [16] designed a smart electronic examination system using a Multi-Agent platform. The system was implemented using the JADE environment;

it was targeted at reducing cheating in exams, and setting questions for examinees according to their performance. The faster the student answers a question the harder the questions become. In this system, agents installed on the PCs use the experience gained from interacting with other agents and examinees to detect cheating and perform its duties.

Comparison: Although the idea of hardening exam questions can help in evaluating the knowledge level of a student, it can also raise issues of unfairness when certain students receive harder questions than others. An alternative medium of assessing students is proposed in this thesis. Student all have a fair and equal chance of answering similar questions in the exams. In this system, the knowledge level of students is not determined through the hardening of questions but by analysing the student's performance during practise exams and formal exams. With the help of agents the performace of students can be easily calculated and their areas of weakness or strength is displayed. With this the students can know what level thay are in and the areas they need to improve in.

- Khan and Abbas implemented a PHP and MySQL based online examination system with power failure handling and Dropbox capabilities [5]. The system enables students resume an examination from where they stopped before power failure and, solves the problems of online examinations in countries with frequent power cuts. The system also creates flexibility in the way questions are answered by introducing Dropbox features which enable students save difficult questions in Dropbox and answer them at a later time.

Comparison: The system provides a user-friendly and convenient environment for students to take exams but since some of its features is dependent on Dropbox it is open to various problems if any Dropbox failure or delay occurs. In this thesis Dropbox independent flexibility in answering questions is provided. Students can also skip difficult or ambiguous question and answer them at a later time but this is not done with the help of Dropbox but through tactical database structuring (specific tables have been created to determine answered and unanswered questions) and specific database queries made by agents on the system's server. The related work mentioned also does not have features that can introduce smartness or autonomy. In this thesis agent-technologies is used in order to provide opportunities through which intelligence can be introduced into the system (details on intelligence is further described in section 3.2).

- Using the J2EE platform, researchers Xiaoyu and Yunhao presented a research and implementation of a web-online English testing system [4]. With the same platform researcher Lu and Hu also designed and implemented an online examination system [10]. Both researches basically divided the operations of their system into modules e.g. Exam management and Scores management. Each module has specific duties they perform and the system was programmed according to these specifications.

  Comparison: Although these systems provide a good alternative to paper-based systems, there is still room for further improvements by the implementation of additional features and system functions e.g. providing hurry up reminders during

exams, displaying estimated time to finish, providing opportunities to answer ambiguous questions later, setting and taking practise exams, sending requests for extra time or attention to teachers during exams, handling exam requests by teachers, getting useful agents analysis of performance, amongst others. These features listed are provided in this thesis.

- Another online examination system based on B/S (Browser/Server) structure was developed by Li and Wu [8]. The system offers various features that help both the teachers and students improve their skills. Students are given opportunities to test themselves and be involved in various academic contests and activities that build up their knowledge. The system also enhances teaching skills by providing techniques for analysing students, exam papers and questions.

  Comparison: The ideas proposed in this system are creative, but on the other hand, introducing a certain level of intelligence by assigning functions of the system's modules to an intelligent software or agent network will greatly increase the efficiency of the system. This thesis looks into creating a system where the bulky and complex tasks of a system is broken down into simpler subtasks handled by agents in a network. This not only makes the system easier to manage, debug and maintain but also opens door for various smart features to be added into the system.

1.5 Aims and Motivation

In this section I will discuss what this thesis hopes to achieve. The aim of this thesis is to:

1. Provide solutions to the limitations of paper-based examination systems.
2. To create an online examination system that is geared towards enhancing the skills of students and teachers.
3. To offer a flexible, smart and convenient alternative for conducting exams and assessing students.
4. To provide students with a user-friendly, fun and helpful guide for knowledge enhancement.
5. To improve the quality of teaching and learning, and the general standard of education by enhancing academic evaluation methods.

This thesis is motivated by the need to satisfy the aims stated and the desire to offer better solutions that will make a positive impact on education in the future.

Chapter 2

INTELLIGENT AGENTS AND MULTI-AGENT SYSTEMS

2.1 What is an Agent?

Over the years there have been varyıng definitions for an agent; this due to the fact that the different characteristics of an agent can vary in its importance in different fields or domains [17]. Agents are generally described as computational entities, software components or computer systems that are capable of performing autonomous actions on behalf of their users [2], [12], [17], [14]. Due to their independent and social abilities, agents have been used in wide variety of complex applications [2], [13].

2.1.1 Structure and Characteristics of Agents

Agents perform actions according to the information they perceive from their environment [2], [17]. With the use of physical sensors (used by agents situated in the real world) or software sensors (used by software agents), agents gather information from their environment; they use this information to perform certain actions on their environment with the help of effectors or actuators [17], [18], [19]. The general aim of agents is to satisfy certain goals or objectives they are designed to achieve [2], [17], [18].

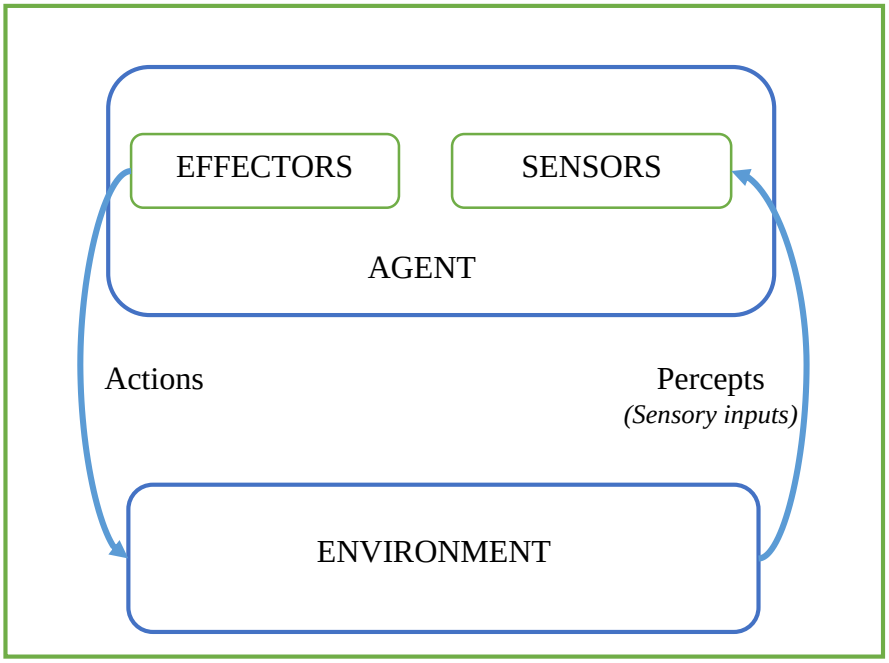


Figure 2.1. Structure of an agent [17]

Intelligent agents exhibit certain characteristics; they are reactive, autonomous, proactive, social, and cognitive, amongst others.

Autonomous: Agents are able to operate without the guidance or intervention of human beings [12], [14]. They are able to take control over their actions and take decisive steps to achieve their goals [14], [20].

Reactive: Agents have the ability to perceive their environment and respond promptly to the changes that occur in their environment [17], [14].

Proactive: Agents do not only act reactively in response to their environment but can also exhibit proactive behaviour i.e. they are able to take the initiative in their quest to satisfy their goals rather than simply responding to their environment [12], [17], [14], [20].

Social: Agents are social entities that are able to interact with other agents or humans in order to accomplish their goals [17], [14], [20]. Agents can communicate with the use of an agent communication language which will be discussed further as we go on [14].

Other characteristics of agents include flexibility, mobility, rationality, learning capabilities [2], [14] etc. All these characteristics assists agents in achieving their goals which is the main aim of an agent.

2.2 Agent Types

Intelligent agents can be classified into various types. Some of them are:

Local or Interface Agents: This type of agent can only access the local resources in its environment and serves mainly as an assistant to users [12], [13]. Local agents help to simplify the complex tasks of users [12], [13]. They have the ability to learn about a user's mannerisms or preferences through observations and can act on behalf of the users [12]. They can also advise, train or teach users in various ways [13].

Networked Agents: This agent makes use of both local and remote resources unlike local agents [13]. Apart from providing services to the user, a networked agent makes use of services available on the network to perform its duties on behalf of users [13]. One of the main uses of networked agents is for information retrieval since the agents can access a wide range of information in a network [12], [13].

DAI (Distributed Artificial Intelligence based) based or collaborative Agents: This type of agent unlike the previously defined ones have the ability to communicate

with one another [12], [13]. A DAI-based agent collaborates or works hand in hand with other agents to perform tasks on behalf of a user [12]. Using this interconnected form of communication, agents are able to assist one another or jointly take actions with the aim of achieving their goals [13]. This type of system was used in the implementation of this thesis and will be described in further details as we go on.

Mobile Agents: Mobile agents are intelligent agents that are capable of migrating from one computer to another. They are capable of stopping execution on one PC and transporting themselves to another PC to continue execution. The mobility of agents is very useful in large computer networks with a wide range of sophisticated services. They help in reducing network load and they also solve host problems e.g. an agent can migrate from a faulty or corrupt system to another one and continue execution.

Other types of agent classifications are learning agents, Utility-based agents, Emotional agents, Simple reflex agents, Goal-based agents, Model-based agents etc.

2.3 Agent Environment Types

As mentioned previously, agents gather information from the environment they perceive and also perform actions on their environment. There are various environments that an agent can work in. Some of them are given below.

Deterministic vs. Non-deterministic: An environment is deterministic if the next state of the environment or the resulting effect of actions taken on an environment can be determined by its current state i.e. there is no uncertainty about the state of the environment after an action has been performed on it [17], [19]. A non-deterministic

environment is the opposite; there are uncertainties about the state of the environment and the resulting effect cannot be determined by its current state.

Accessible vs. Inaccessible: An environment is accessible if an agent can obtain all the information about the environment [17], [19]. If agent's sensors can perceive or gain access to all relevant parts of its environment then the environment is said to be accessible to the agent [19]; an inaccessible environment on the other hand is vice versa.

Static vs. Dynamic: An environment is said to be static if and only if it remains unchanged except by actions performed on it by an agent [17]. In a dynamic environment, other factors other than the actions of an agent can cause changes in the environment, and these changes can sometimes be beyond the control of the agent [17].

Episodic vs. Non-episodic: An episodic environment is an environment in which an agent's experience is divided into episodes [19]. Each episode is independent of the previous ones [17], [19]. Agents can independently perceive and act in an episode without being affected by previous episodes [19]. The main focus of agents in this environment is its current episode [17], [19]. Non-episodic environment are not so; previous actions can affect future ones.

Discrete vs. Continuous: An environment is said to be discrete if it has a fixed number of possible precepts and actions in it; a continuous environment is the opposite (no fixed number or percepts and actions) [17], [19].

2.4 Application of Intelligent Agents in the Real World

Agents have been applied in many areas in the real world. They can be used in information retrieval, information filtering, smart messaging, ecommerce, email management, network management, system management, business operations, telecommunications and entertainment [12], [13], [21]. Agents have been used in implementing a wide range of applications such as Personal assistants, Advisory assistants, teaching assistants or eLearning assistants, smart mail boxes, intelligent user interfaces, meeting scheduler, customer help desk, personal shopping assistants, mobile applications and so much more [12], [13], [21], [22], [23].

2.5 What are Multi-agent Systems?

A multi-agent system (MAS) is an interconnected network of intelligent agents that are capable of interacting with one another [24], [25]. The agents work together to achieve common goals and are capable of performing distributed tasks concurrently [26]. One of the main aims of a MAS is to find solutions to complex system problems and to handle tasks that are beyond the ability of a single agent [27], [28].

2.5.1 Structure of Multi-agent Systems

Figure 2.2 shows a multi-agent system with agents communicating with one another. The agents observe things in their environment and make use of or perform actions on the resources in their environment.

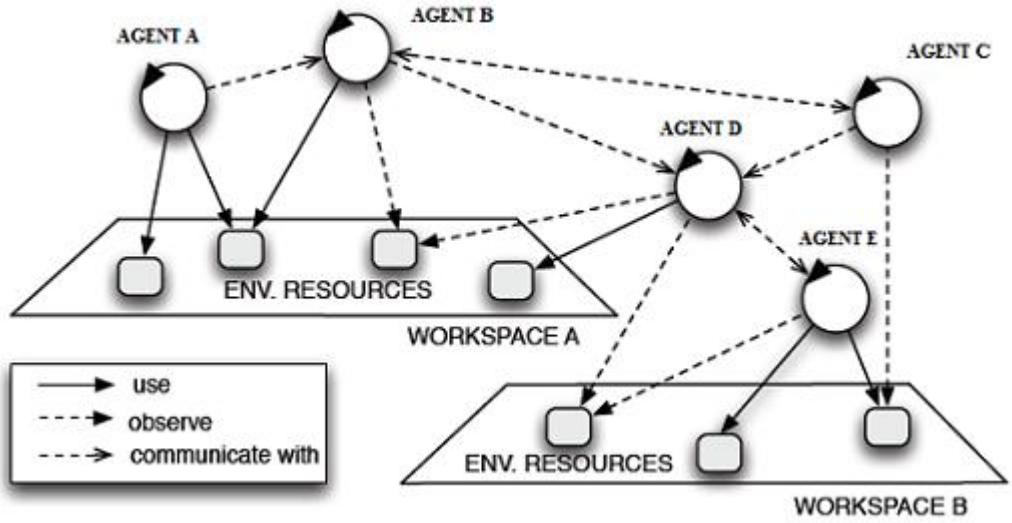


Figure 2.2. Multi-agent Systems [29]

2.5.2 **Applications of Multi-agent Systems**

Due to their ability to solve complex problems, multi-agent systems have been applied in so many areas, some of which are:

- Aircraft Management or maintenance [25]
- Distributed vehicle monitoring [30]
- Military defence systems [25]
- Wireless communications [25]
- Telecommunications [14]
- E-learning [31]
- Online Examination Systems [16]
- Multi-robotic systems [14]
- Health care management [14]

2.5.3 **Advantages of Multi-Agents Systems**

Multi-agent systems have a wide range of advantages:

- They are capable of handling the problems of large complex distributed systems that may be difficult for humans or a single agent to handle [25], [27], [28].

- Due to the collaborative nature of MAS, computerized systems in which multi-agent systems are incorporated, significantly improve in performance especially in the area of flexibility, efficiency, reliability, extensibility, robustness amongst others [25].

- MAS are efficient in filtering, retrieving and organising information from widely distributed sources [25].

- In MAS, resources and information are efficiently distributed over a network of agents i.e. Agents do not experience limitations in accessing the necessary resources that they need in the system [25].

Chapter 3

INTELLIGENT MULTI-AGENT ONLINE EXAMINATION SYSTEM

3.1 System Definition

Intelligent multi-agent online examination system as the name implies is an examination system handled by multiple agents in a network. Each agent in the system has duties that it performs and can communicate with other agents to get the information that it needs to accomplish its personal goals. Although each agent has individual goals it aims at satisfying, agents as a whole also have a common goal, which is to improve the efficiency, smartness and flexibility of the examination system. The system was implemented for use in a university environment but can be extended to other areas as well.

3.2 The Term 'Intelligent'

The attribute or characteristics that make an agent 'intelligent' varies across field and domains. In general an agent is deemed intelligent if it portrays the following characteristics:

- **Reactive:** The ability to perceive its environment and respond promptly to the changes that occur in its environment [17], [32], [33].
- **Proactive:** Ability to take the initiative to satisfy its goals (goal-oriented) [17], [32], [33].

- **Autonomous:** Ability to perform tasks independently without the intervention of humans [32].
- **Collaborative or social:** ability to work with and interact with other agents or humans [17], [32], [33].
- **Ability to learn/Adaption:** Ability to learn from experience and improve its performance over time [32], [33].

These characteristics amongst others, (e.g. mobility, rationality) are the terms used by many to determine if an agent is intelligent or not. In this thesis however, the agents do not exhibit all the characteristics specified above. The 'intelligence' of the OES agents in this thesis is restricted to the following characteristics:

- They perform various tasks on behalf of users (autonomous)
- They are goal-oriented i.e. they are capable of satisfying goals or duties they are designed to accomplish.
- They have the ability to communicate with other agents in their quest to satisfy their goals (communicative and collaborative).

It is on the basis of the attributes listed above that the OES agents are classified as 'intelligent'.

3.3 Why Multi-agent Systems?

The question 'why multi-agent systems over other alternatives?' can be answered with the follows points:

- **Parallelism:** The tasks of multi-agent based systems are distributed to various agents that can perform their work in parallel [34]. This helps in improving the speed and performance of the system [34].

- **Fault tolerance:** Multi-agent systems are fault tolerant [34]. If the task and duties of a system are adequately distributed among agents, failure that may occur in one or more agents may not affect the entire system [34].
- **Simplified programming:** Multi-agent systems are generally modular in nature i.e. the system functions or operations of the system are divided into different modules [34]. Due to its modular nature the system is generally easier to program since the large complex task of the system is broken down into simpler parts that are assigned to different agents [34].
- **Scalability:** Another good feature of multi-agent systems is its scalability. Due to its modular nature, it's easier to make changes to the system [34] i.e. new agents or features can be easily added to the system whenever necessary [34].
- **Intelligence:** Using multi-agent system opens up opportunities for intelligence to be introduced into a system [34]. Agents on the system are able to perform certain tasks on behalf of users; they are able to reason and make decisions on how their goals can be achieved [34].
- Multi-agent systems are ideal for companies or organizations who need a system that can satisfy the different goals or objectives of the diverse departments in the firm [34].

3.4 System Database

All the information needed in running the examination system is stored in a database called "OnlineExamination". The MySQL Workbench environment was used in creating the database and its tables. The tables used the system are as follows:

1. **Examination table:** Used for storing information about an exam. The table contains the following fields: the examination number, exam type, exam date, exam points, course id, duration, and semester.

Examination_No	Course_ID	Exam_Date	Duration	ExamType	Semester	Exam_points
1	Math101	2015-02-21 1...	10	Midterm	Spring 2014	40
2	Math101	2015-01-10 1...	100	Final	Spring 2014	60
3	Eng101	2015-01-10 1...	100	Midterm	Spring 2014	40
4	Eng101	2015-01-10 1...	100	Final	Spring 2014	60

Figure 3.1. Examination table

2. **Active_examination table:** Stores active exams (exam currently being taken). When the date an time of an exam is reached the exams are activated and stored in the Active_examination table. This table contains the following fields: examination number and course id.

Examination_No	Course_ID
3	Eng101
1	Math101

Figure 3.2. Active_examination table

3. **Inactive_examination table:** Stores deactivated exams (exam already taken). When the duration time of an exam elapses, the exam is deactivated and inserted into the inactive_exam table. This table contains the following fields: examination number and course id.

Examination_No	Course_ID
3	Eng101
1	Math101

Figure 3.3. Inactive_exam table

4. **Administration table:** Used for storing information about teacher. This information is used for authenticating the teacher during login. The table contains the following fields: admin username, admin password, role, admin name, and admin surname.

Admin_username	Admin_Password	Role	Admin_Name	Admin_surname
adm111	m111	Teacher	Admin	Anaekwe
adm222	m222	Teacher	Admin	Abia

Figure 3.4. Administration table

5. **Course table:** Used for storing information about semester courses. The course ID, course name and course credit are all stored in this table. These courses are assigned to the appropriate students and teachers during the semester and activities relating to these courses are performed on the system.

Course_ID	Course_Name	Credit
CHEM101	Chemistry	4
CMPE101	Programming	4
ENG101	English	4
MATH101	Mathematics	4
PHY101	Physics	4

Figure 3.5. Course table

6. **Course_improvement table:** Used for storing the improvement percentage of students. As students take practise exams on the system their improvement percentage is stored in the Course_improvement table and displayed to students when necessary. The table contains the following fields: improvement number (No), student ID, improvement (Percentage out of 100%), and course ID.

No	Student_ID	Improvement	Course_ID
1	20080808	10	MATH101
2	20130101	0	MATH101
3	20130202	9	MATH101
4	20130606	40	CHEM101
5	20130101	25	MATH101

Figure 3.6. Course_improvement table

7. **Courses_taken table:** Used for storing courses taken by students during the semester. This table displays the courses assigned to students in a particular semester. The table contains the following fields: student ID and course ID.

Student_ID	Course_ID
20080808	CHEM101
20080808	CMPE101
20080808	ENG101
20080808	MATH101
20080808	PHY101

Figure 3.7. Courses_taken table

8. **Exam_question_answer table and Options table:** Used for Storing exam questions, answers and options. When teachers set exam questions the questions and options are automatically inserted into the Exam_question_answer table and Options table. These questions are later displayed to students during an exam. The Exam_question_answer table contains the following fields: question ID, examination number, question, answer, points, category. The Options table contains the following fields: question ID, examination number, and options A,B,C,D,E respectively.

Question_ID	Examination_No	Question	Answer	Points	Category
1	1	What is -3 - -5?	E	2	Subtraction
2	1	What is 3 * 35?	A	4	Multiplication
3	1	What is -2 * 25?	E	4	Multiplication
4	1	What is 2.5 * 8.1 ?	B	4	Multiplication
5	1	What is -4 * -5 ?	C	4	Multiplication
6	1	What is 1.5 * 1.5 ?	A	4	Multiplication
7	1	What is X given equation 2X = 4 ?	C	6	Algebra

Figure 3.8. Exam_question_answer table

Question_ID	Examination_No	OptionA	OptionB	OptionC	OptionD	OptionE
1	1	8	-2	-8	7	2
2	1	105	335	65	350	353
3	1	23	50	-23	-27	-50
4	1	21	20.25	20	202.5	2.025
5	1	-20	-9	20	9	-15
6	1	2.25	3.0	22.5	0	30
7	1	6	8	2	4	0.5
8	1	1.5Y	5Y	6Y	-1.5Y	5

Figure 3.9. Options table

9. **Exams_taken table:** Stores exam already taken by students. Once an exam has been completed by a student, information about the exam and the student who took the exam is stored in this table. This table contains the followimg fields: student ID and examination number.

Student_ID	Examination_No
20080808	1
20130101	1
20130606	1
20130606	3

Figure 3.10. Exams_taken table

10. **Extra_time table:** Stores the extra time given by teachers in a particular exam. When a teacher decides to give extra time for an exam, the extra time given is inserted into this table with the corresponding course ID, exam type, and time of addition.

Course_ID	ExamType	Extra_time_added	Time_of_Addition
MATH101	Midterm	10	2015-02-21 14:14:00
ENG101	Midterm	5	2015-01-10 14:00:00

Figure 3.11. Extra_time table

11. **Selected_log table:** Stores exam questions already viewed/attempted by students during an exam. This prevents students from viewing or answering questions previously displayed. The table contains the following fields: examination number, question ID, question number, and student ID.

Examination_No	Question_ID	Question_No	Student_ID
1	1	1	20080808
1	2	2	20080808
1	3	4	20080808
1	4	4	20080808

Figure 3.12. Selected_log table

12. **Final_select_log table:** Students are given the opportunity to skip ambiguous questions and answer them later. After all questions have been attempted, a student can start answering his/her previously unanswered questions. Once these questions are answered they are immediately inserted into the Final_select_log table. This helps to determine the answered/unanswered questions of a student i.e. questions whose ids are not found in this table can still be answered by students. The table contains the following fields: examination number, question ID, question number, and student ID.

Examination_No	Question_ID	Question_No	Student_ID
1	6	6	20080808
1	10	10	20080808
1	18	18	20080808
1	20	20	20080808

Figure 3.13. Final_select_log table

13. **Sim_selected_log table:** Stores exam questions already viewed/attempted by students during a simulation exam. This prevents students from viewing or answering questions previously displayed. The table contains the following fields: simulation examination number, simulation question ID, and student ID.

SimExam_No	SimQuestion_ID	Student_ID
SimMath101A	1	20130404
SimMath101A	2	20130404
SimMath101A	3	20130404
SimMath101A	4	20130404

Figure 3.14. Sim_selected_log table

14. **Sim_final_select_log table:**. Students are given the opportunity to skip ambiguous questions and answer them later. After all questions have been attempted, a student can start answering his/her previously unanswered questions. Once these questions are answered they are immediately inserted into the Sim_final_select_log table. This helps to determine the answered/unanswered questions of a student i.e. questions whose ids are not found in this table can still be answered by students. The table contains the following fields: simulation examination number, simulation question ID, and student ID.

SimExam_No	SimQuestion_ID	Student_ID
SimMath101A	2	20130404
SimMath101A	17	20130404
SimMath101A	19	20130404

Figure 3.15. Sim_final_select_log table

15. **Sim_student_ans table:** Stores answers selected by students in a simulation exam and is used for calculating the student's grades. The table contains the following fields: simulation examination number, simulation question ID, student answer, category, and student ID.

SimExam_No	Student_ID	SimQuestion_ID	Student_Sim_Ans	Category
SimMath101A	20080808	1	A	Addition
SimMath101A	20080808	2	B	Multiplication
SimMath101A	20080808	3	E	Algebra
SimMath101A	20080808	4	C	Subtraction

Figure 3.16. Sim_student_ans table

16. **Sim_student_category_grade table:** Stores the grades (practice exams grades) of students in the different topics of a course and is used for displaying the topics a student is good or bad in. The table contains the following fields: student ID, simulation exam number, category, category grade, number of correct answers in a category (No_correct_ans), and course ID.

Student_ID	SimExam_No	Category	Grade	No_correct_ans	Course_ID
20080808	SimMath101A	Addition	80	2	MATH101
20080808	SimMath101A	Multiplication	75	3	MATH101
20080808	SimMath101A	Algebra	50	5	MATH101
20080808	SimMath101A	Subtraction	100	0	MATH101

Figure 3.17. Sim_student_category_grade table

17. **Sim_student_grade table:** Used for storing the practice exam grades of students in various courses. The grades are displayed to students after each exam. The table contains the following fields: student ID, simulation exam number, student simulation exam points, grade, and course ID.

Student_ID	SimExam_No	Sim_Student_exam_points	Grade	Course_ID
20080808	SimMath101A	32	80	MATH101
20130303	SimMath101A	30	75	MATH101
20130404	SimMath101A	20	50	MATH101
20130606	SimMath101A	40	100	MATH101

Figure 3.18. Sim_student_grade table

18. **Simexam_q_a table and Simoptions table:** Used for Storing exam questions, answers and options. When teachers set practise exam questions, the questions and options are automatically inserted into the Simexam_q_a table and Simoptions table. These questions are later displayed to students during a practise exam. The Simexam_q_a table contains the following fields: simulation question ID, simulation examination number, simulation question, simulation answer, points, category. The Simoptions table contains the following fields: simulation question ID, simulation examination number, and options A,B,C,D,E respectively.

SimExam_No	SimQuestion_ID	SimQuestion	SimAnswer	Points	Category
SimMath101A	1	What is 1 + 1?	A	2	Addition
SimMath101A	2	What is 3 * 2?	B	4	Multiplication
SimMath101A	3	Given the equ...	E	6	Algebra
SimMath101A	4	What is 2 - 5?	C	2	Subtraction

Figure 3.19. Simexam_q_a table

SimQuestion_ID	SimExam_No	OptionA	OptionB	OptionC	OptionD	OptionE
1	SimMath101A	2	4	7	8	10
2	SimMath101A	5	6	10	32	11
3	SimMath101A	16	1	8	3	2
4	SimMath101A	-7	7	-3	3	2

Figure 3.20. Simoptions table

19. **Simulationexam table:** Used for storing information about a simulation or practise exam. The table contains the following fields: simulation examination number, course id, duration, and semester.

SimExam_No	Course_ID	Duration	Semester
SimEng101A	Eng101	10	Fall
SimMath101A	Math101	10	Fall

Figure 3.21. Simulationexam table

20. **Student table:** Used for storing information about students. The student's ID, name, surname, department, password are all stored in this table. The information in this table is used for logining in students, assigning courses to students, assigning grades, amongst others.

Student_ID	Student_Name	Student_Surname	Department	Student_Password
20080808	Jane	Anaekwe	Computer Eng...	080808
20130101	Ubong Lois	Abia	Biomedical En...	130101
20130202	Jackie	Strongtower	Marine Engine...	130202
20130303	Blessing	Idehen	Biomedical En...	130303
20130404	Iveren	Kajoh	Computer Eng...	130404
20130505	Dooter	Kajoh	Industrial Engi...	130505
20130606	Cyril	Ede	Computer Eng...	130606

Figure 3.22. Student table

21. **Student_answers table:** Stores answers selected by students in an exam and is used for calculating the student's grades. The table contains the following fields: examination number, question ID, student answer, category, and student ID.

Examination_No	Student_ID	Question_ID	Student_Answer	Category
1	20130202	1	E	Subtraction
1	20130202	2	A	Multiplication
1	20130202	3	E	Multiplication
1	20130202	4	B	Multiplication

Figure 3.23. Student_answers table

22. **Student_category_grade table:** Stores the exam grades of students in the different topics of a course and is used for displaying the topics a student is good or bad in. The table contains the following fields: student ID, exam number, category, category grade, and the number of correct answers in a category (No_correct_ans).

Student_ID	Examination_No	Category	Grade	No_correct_ans
20130505	1	Subtraction	90	2
20130505	1	Multiplication	80	3
20130505	1	Algebra	70	4
20130505	1	Addition	100	0

Figure 3.24. Student_category_grade table

23. **Student_grade table:** Used for storing the exam grades of students in various courses. The grades are displayed to students after each exam. The table contains the following fields: student ID, exam number, student exam points, grade, and exam type.

Student_ID	Examination_No	Grade	ExamType	Student_exam_points
20080808	1	95	Midtem	38
20080808	1	100	Final	60

Figure 3.25. Student_grade table

24. **Student_requests table:** Used for storing the exam requests made by students in an exam. Students can either make a request for extra time or for the teacher's attention which is stored in the 'request' column of the table. The table contains the following fields: student ID, student name, student surname, course ID, request, time of request, and exam type.

Student_ID	Student_Name	Student_Surname	Course_ID	Request	Time_of_Request	ExamType
20080808	Jane	Anaekwe	MATH101	Extra time	2014-12-03 14:12:00	Midterm
20080808	Jane	Anaekwe	MATH101	Attention	2014-12-03 14:13:00	Midterm

Figure 3.26. Student_requests table

25. **Teacher_courses table:** Used for storing courses assign to teachers for a particular semester. Activities related to these courses are performed by teachers on the system. The table contains the following fields: admin username, course ID.

Admin_username	Course_ID
adm111	MATH101
adm111	PHY101
adm222	CHEM101

Figure 3.27. Teacher_courses table

Figure 3.28 shows the ER diagram of the database with the respective relationships and connections between tables.

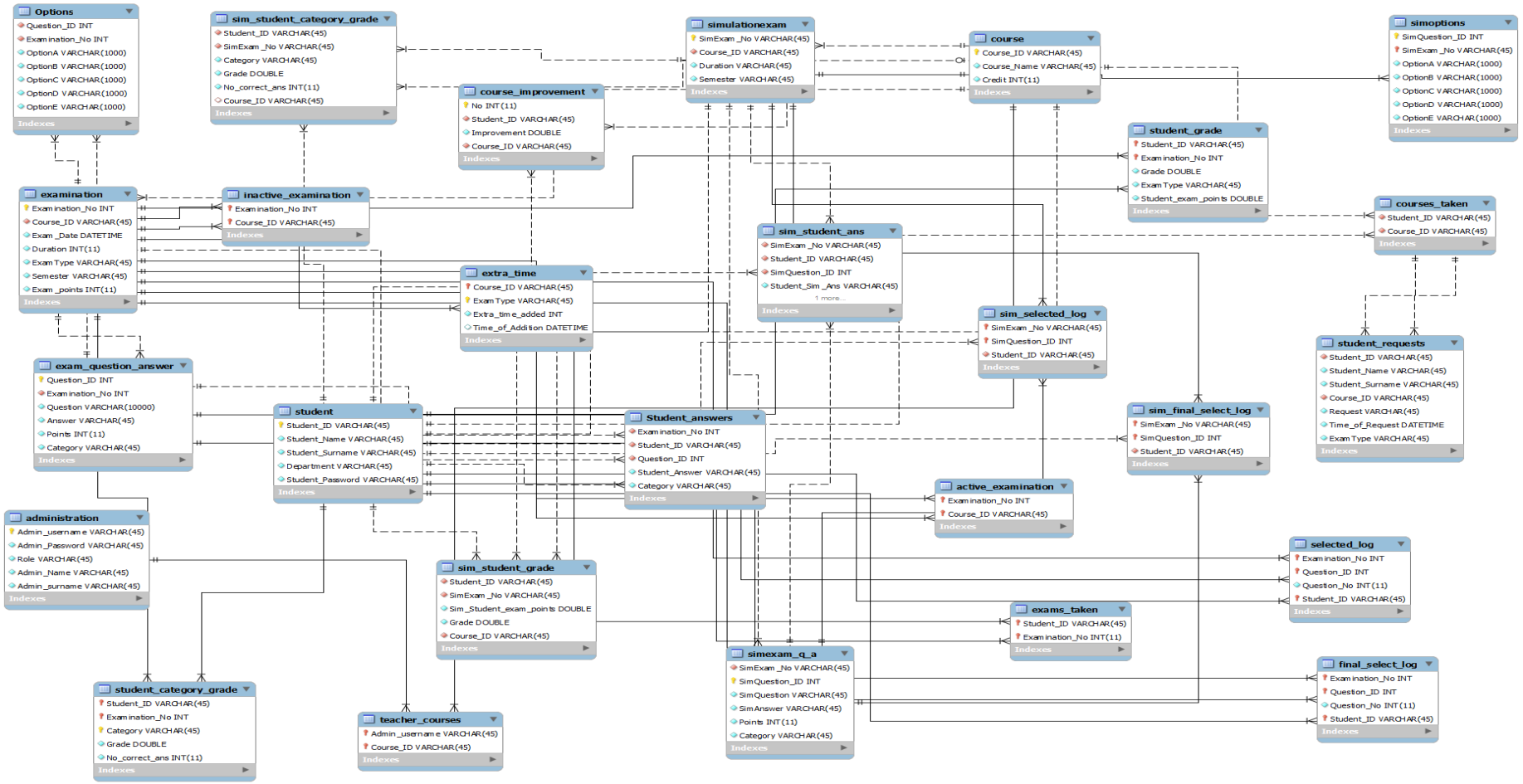


Figure 3.28. ER Diagram

3.5 System Structure

The multi-agent examination system is split into two parts: the user side and the server side. There are five agents located on each side of the system. On the server side we have the following agents: Login agent, Info extract agent, Admin info extract agent, Exam server agent and Simulation agent. On the user's side we have the following: Login user agent, student assistant agent, teacher assistant agent, exam monitor agent, and simulation monitor agent. Each agent has roles to play on the system and will be discussed later in section 3.5.2.

Figure 3.29 shows the structure of the system and user-server communication between agents on the user side and agents on the server side.

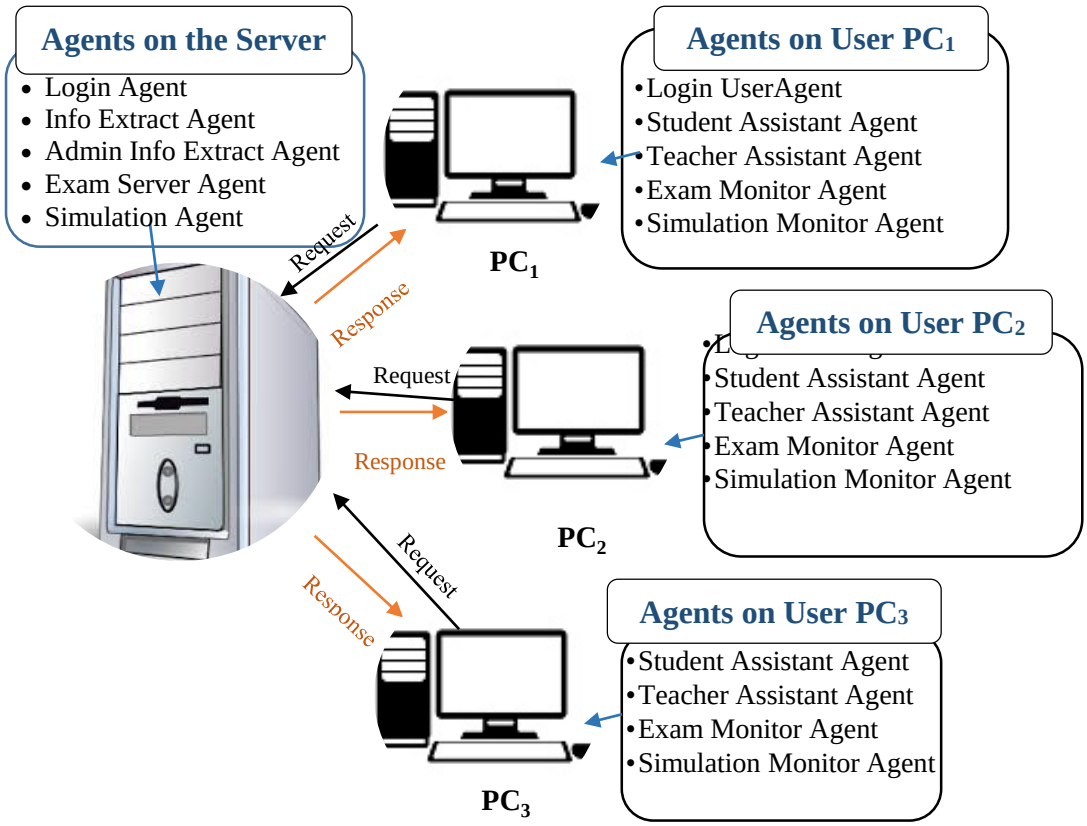


Figure 3.29. System Structure: User-Server relationship

3.5.1 **Modules of the System**

The system is divided into various parts. The three major parts of the system are:

Teacher activity module: This is the part that deals with the operations performed by teachers.

Student activity module: This part deals with the activities done by students.

Server activity module: This deals with the operation done by the server.

A detailed description of the operations of each module is given in sections 3.5.2 and 3.5.3. Figure 3.30 shows the various parts of the system. To increase the efficiency of the system, agents were assigned to each module and perform specific duties in them.

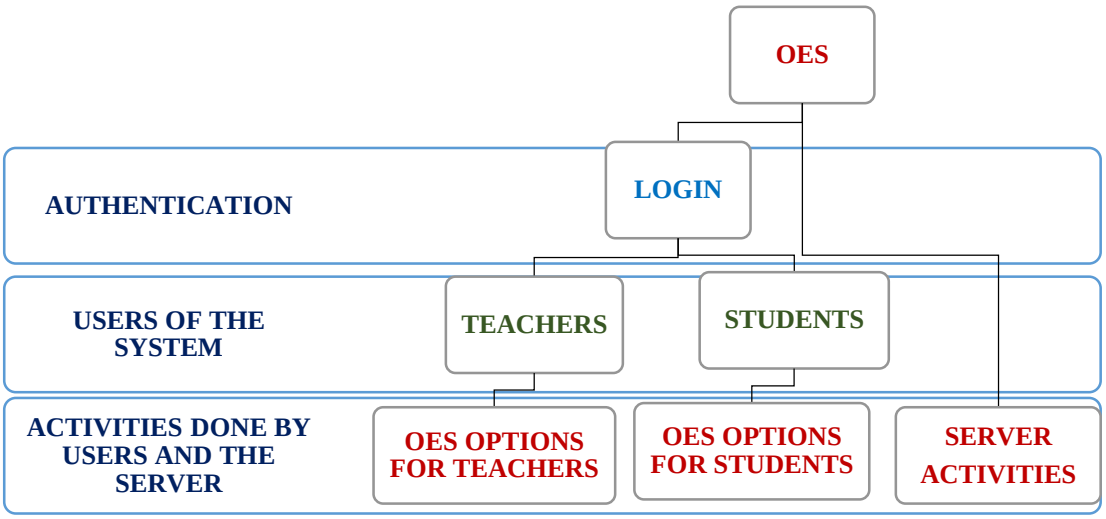


Figure 3.30. Modules of the System

3.5.2 **Roles of the System Users**

As shown previously the two major users of the multi-agent OES are teachers and students. There are several activities that users can perform on the multi-agent OES. Figure 3.31 gives a descriptive illustration of the activities of teachers on the system.

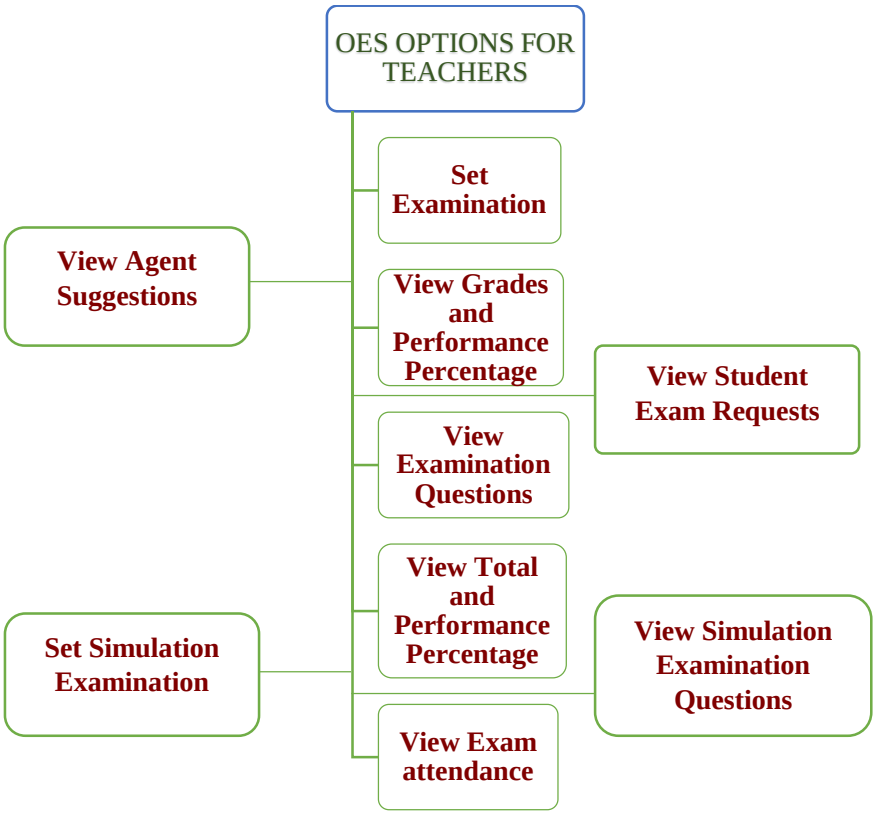


Figure 3.31. Activity of Teachers on the OES

With the help of agents teachers can perform the activities described in Figure 3.31.

- Teachers can view suggestions by agents on which topics students need to improve on in a particular course.
- They can view requests (Extra time requests and requests for teacher's attention) sent by students (via agents) in the exam hall.
- They can set examinations questions or practice questions for students to solve.

- Teachers can view grades, performance percentages, totals and exam scripts all provided to them by the agents on the system.

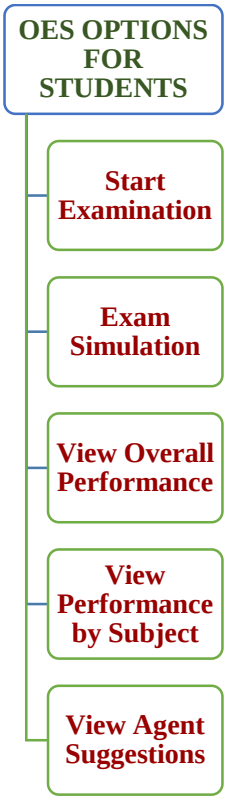


Figure 3.32. Student Activities on the OES

Figure 3.32 give a descriptive illustration of the activities of students on the system. With the help agents students can perform the following activities on the system:

- They can take exams in various courses. During the exam, agents assists students in sending requests for extra time or attention.
- Students can get suggestions from agents on topics they should improve on.
- They can view their grades, totals and performance percentages in various courses.
- Students can also take practice or simulation exams to improve their knowledge in a particular course and also get accustomed to the OES environment.

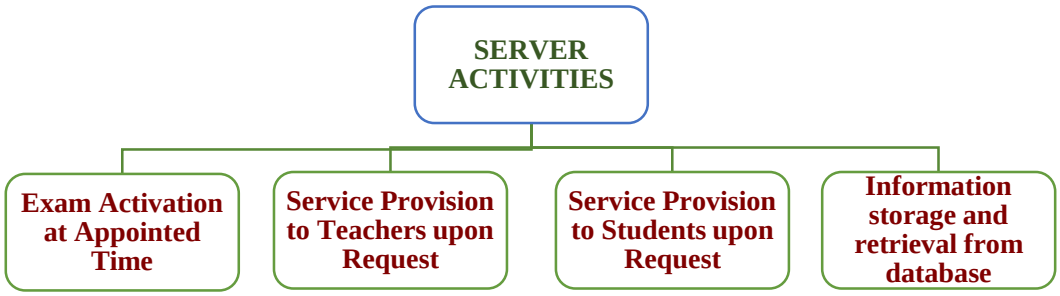


Figure 3.33. Activities of the server

The operations the server performs is a very important part of the system. Without the services provides by the server, the users will be unable to perform their tasks. Vital information needed by the user-side of the system is provided by the server and stored in the server. The server is responsible for the activating exams at the appointed time, without this activation students will be unable to take their exams.

3.5.3 **Roles of Agents on the system**

As mentioned in section 3.5, there are five agents on both the user and server side of the system. In this section the functions of the agent will be described in detail. The functions of agents on the user-side are as follows:

Login user agent: This agent is responsible for authenticating users and providing access into the system. The login agent sends the information entered by users to the login user agent on server who then assists in validating users of the system on the server side.

Exam Monitor agent: Monitors and coordinates exams taken by students. It is responsible for displaying exam questions, remaining time, extra time, hurry up reminders, estimated time to finish, grades, exam scripts and performance statistics to the students. It also send student exam requests (request for attention or extra time) to

the Info extract agent, student answers to the Exam server agent, and deactivates exams on the user-side when the remaining time is exhausted.

Simulation Monitor agent: Monitors and coordinates practice or simulation exams taken by students. It is responsible for displaying exam questions, remaining time, estimated time to finish grades, exam scripts and performance statistics to the students. It also sends student answers to the Simulation agent on the server-side, and deactivates exams on the user-side when the remaining time is exhausted.

Student Assistant agent: Provides assistance to the students with the help of the info extract agent on the server-side. It displays student performance statistics, agent suggestions, student grades and totals to the students.

Teacher Assistant agent: Provides assistance to the teachers with the help of the Admin Info Extract agent on the server-side. It displays student performance statistics, agent suggestions, exam questions, student grades, student total, student exam requests, and student attendance to teachers. This agent also helps teachers in setting exam/practice questions, and providing extra time information entered by the teacher on the user side.

The functions of agents on the server-side is as follows:

Login agent: Using the information provided by the login agent on the user-side the login agent on the server side compares the data sent with the user data stored in the database. If the user information stored in the database corresponds with the one sent, the login agent sends a message to the login user agent, successfully validating the user. If both data do not correspond then the login agent sends an invalid user message.

Admin Info Extract agent: The admin info Extract extracts information about student attendance, student exam requests, agent suggestions, exam questions, student grades, total and performance for use by the teacher assistant agent. It also assists the teacher in inserting extra time and exam questions into the database.

Exam Server agent: This agent works together with the Exam Monitor Agent in handling exams by providing the necessary information and services needed. It sends exam questions and options to the exam monitor agent, sends exam duration and deactivation time, inserts student's exam answers into the database, calculates student grades, displays exam scripts and activates or deactivates an exam at the appropriate time.

Info Extract agent: This agent is responsible for providing information and services to the student assistant agent and the exam monitor agent. The info extract agent extracts information about agent suggestions, student performance and extra time provided by teachers. It also stores information about the requests student made during exams and sends extra time information to the exam monitor agent.

Simulation agent: Works together with the Simulation Monitor Agent in handling exams by providing the necessary information and services needed. The Simulation agent sends the exam questions and options to the Simulation monitor agent, inserts student's exam answers into the database, calculates student grades, displays exam scripts, displays student improvement percentage (increase or decrease) and activates/deactivates an exam at the appropriate time.

3.5.4 Communication between user agents

In this section descriptive illustrations of how agents function/communicate (as explained in section 3.5.3) are shown to give further understanding on how the agents work, and to show the database tables queried by server agents for information retrieval.

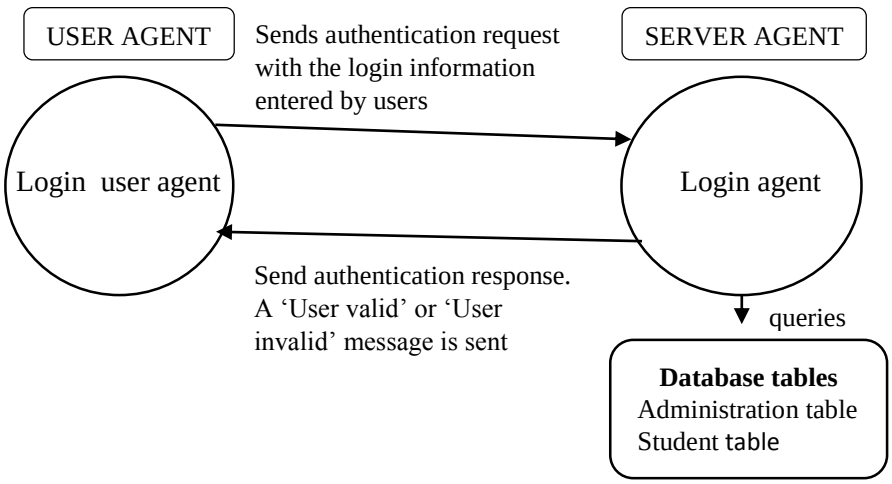


Figure 3.34. Comunication between Login user and Login agent

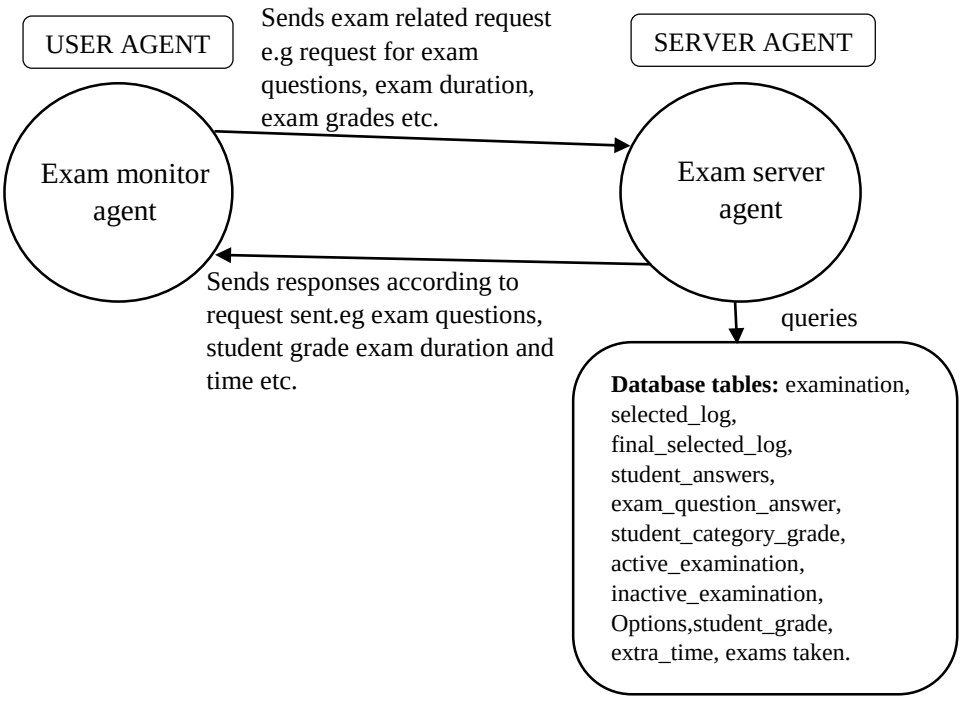


Figure 3.35. Comunication between Exam monitor and Exam server agent

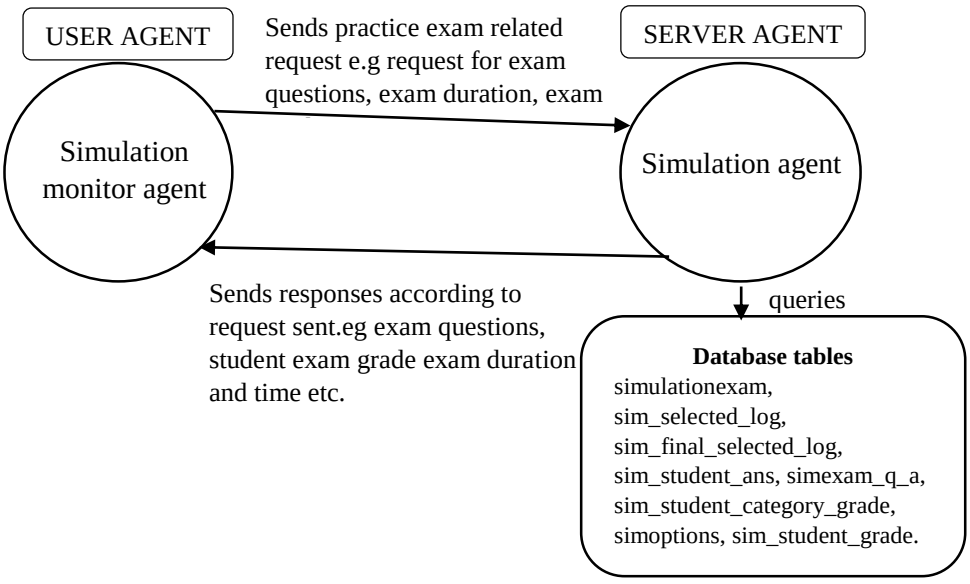


Figure 3.36. Comunication between Simulation monitor and Simulation agent

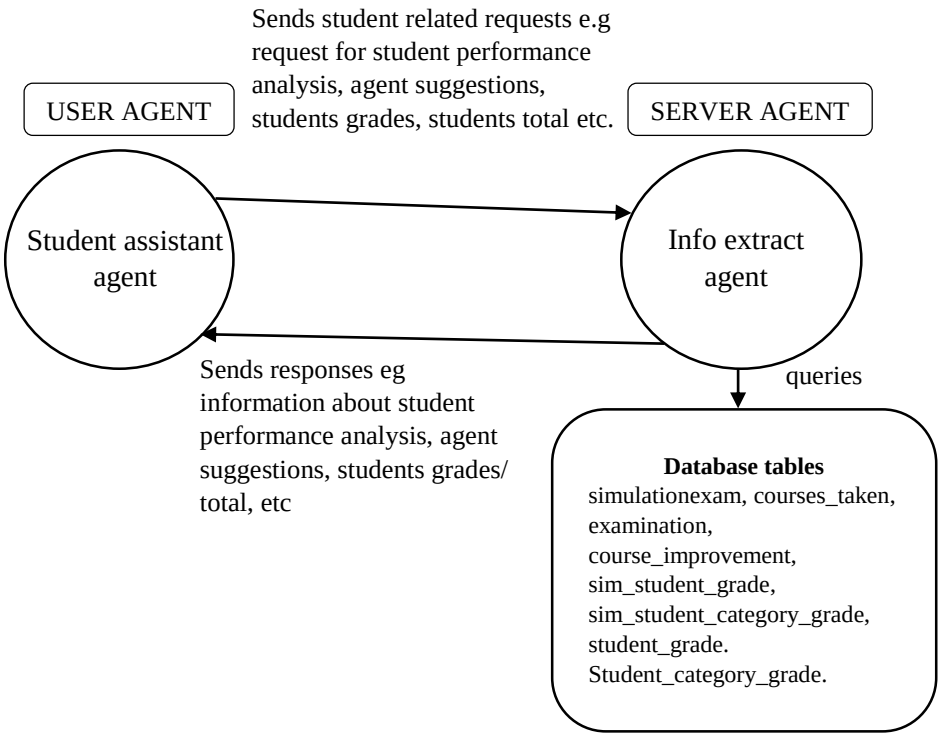


Figure 3.37. Comunication between Student assistant and Info extract agent

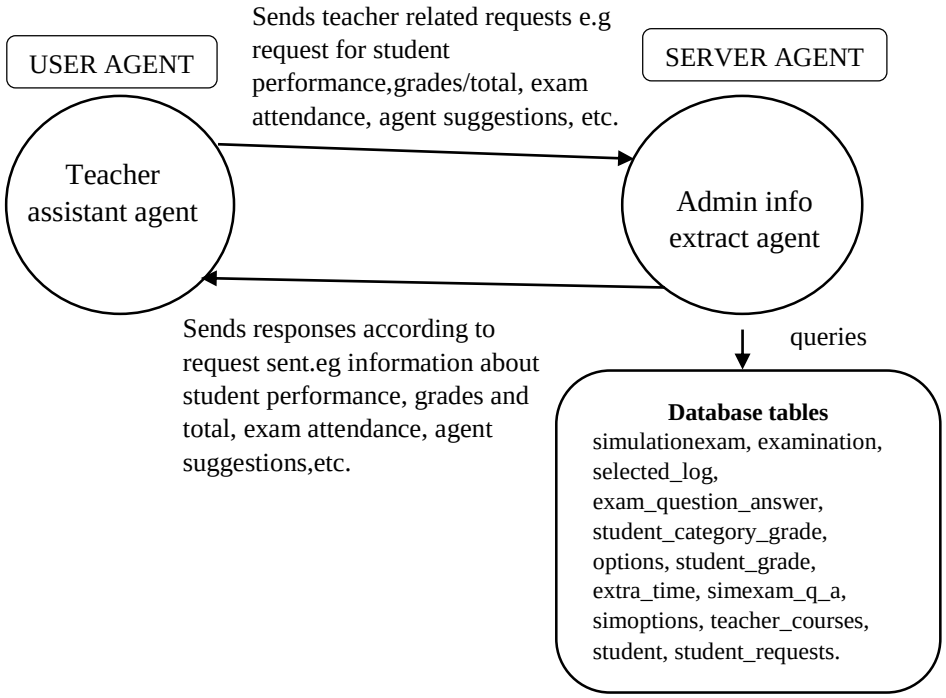


Figure 3.38. Comunication between Teacher assistant and Admin info agent

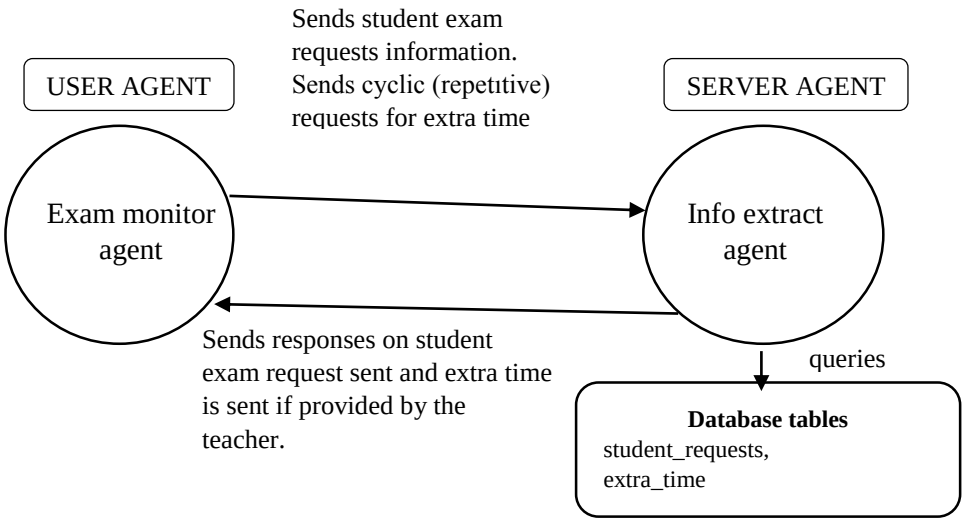


Figure 3.39. Comunication between Exam monitor and Info extract agent

3.6 **How the System Works**

In this section an illustrative step by step approach will be taken to explain how the system works and to describe the features of the system.

The Login portal: Users login to the system using their username and password. If they are valid users, access to the start page of the system is given to them. Access is denied to invalid users or users who enter incorrect information. Figure 3.40 shows the login portal.

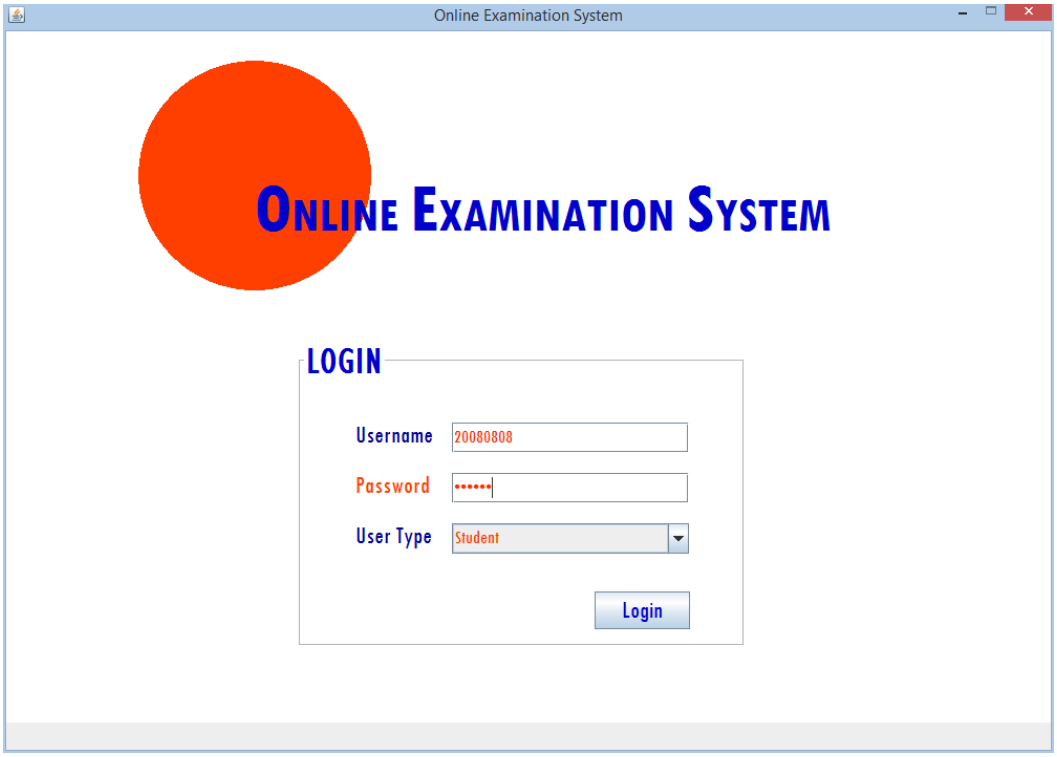


Figure 3.40. Login Portal

Teacher's Start Page: Upon successful login by teachers, the teacher's start page is displayed. Teachers can see the courses they are handle and can select operations they want to perform from the "Options" dropdown button. Figure 3.41 shows the teachers start page.

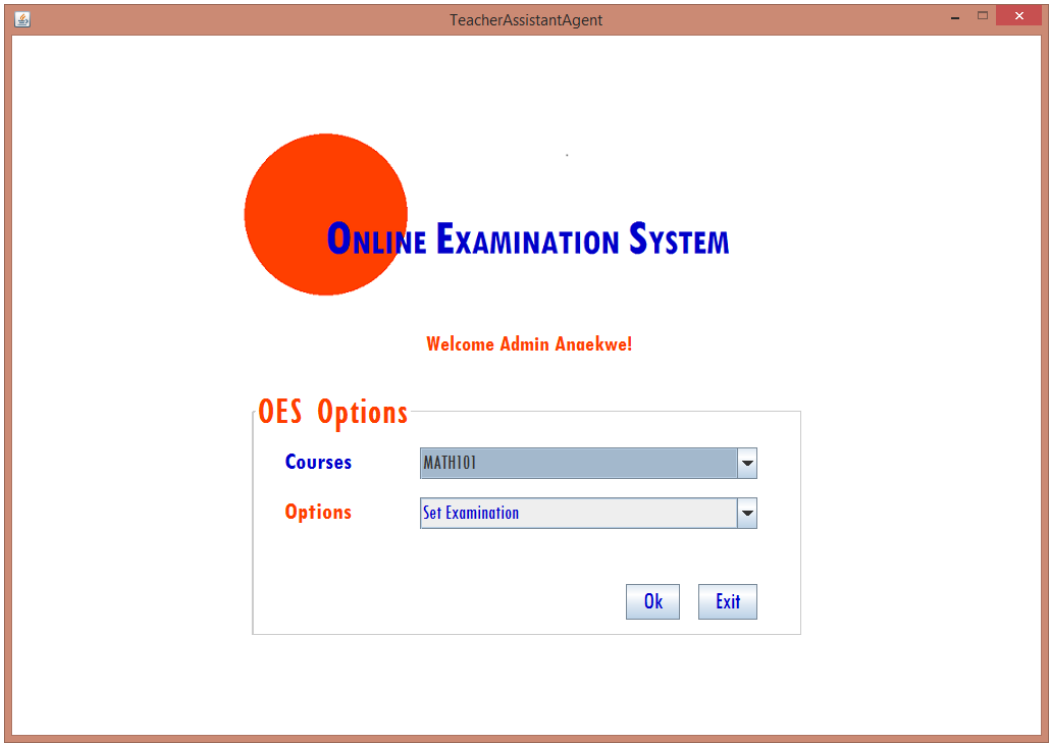


Figure 3.41. Teacher Start Page

Setting Examination questions: Teachers can set exam questions using the "Set Examination Questions Page". Once the necessary data is inserted, the "Set Questions" button is clicked; the questions are automatically inserted into the database. Figure 3.42 show the design structure of the "Set Examination Questions Page".

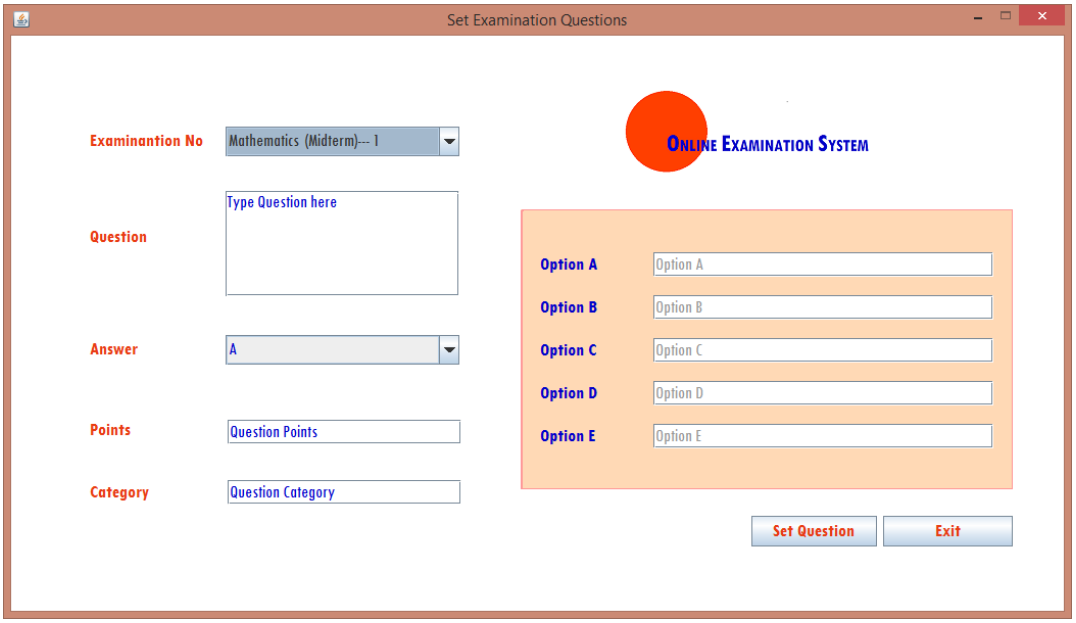


Figure 3.42. Set Examination Questions Page

Viewing Grades and Performance Percentage: Teachers can view student grades and performance percentage by selecting "View student's Grades and Performance" from the start page dropdown options. The teacher assistant agent also displays a comment on what its thinks about the performance of the students at the bottom of the page. Figure 3.43 shows the student's grade and performance page.

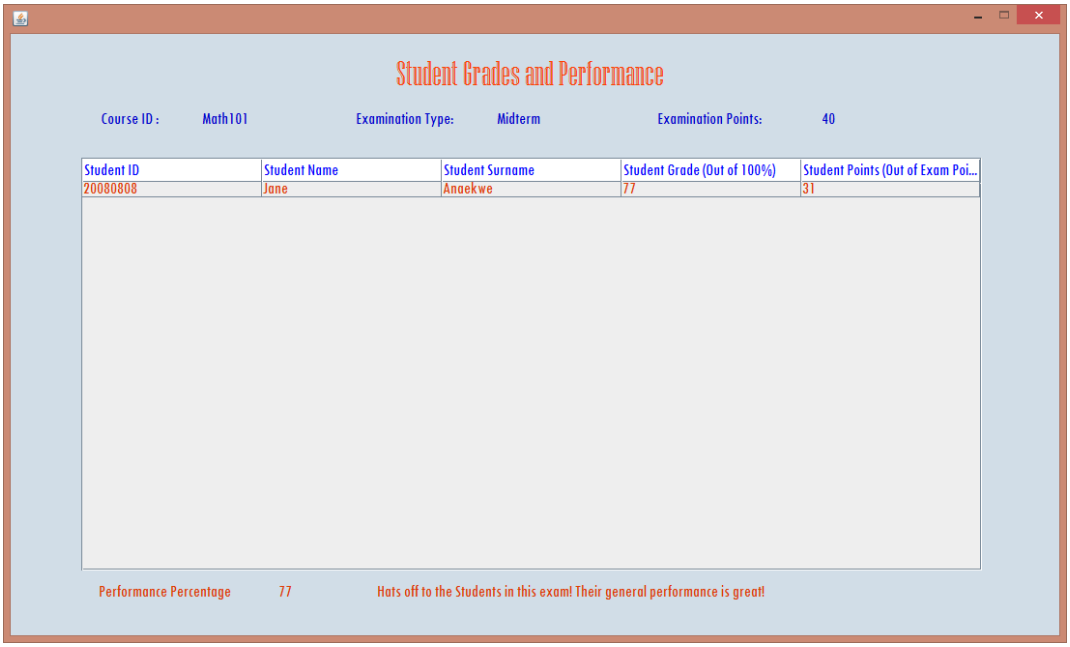
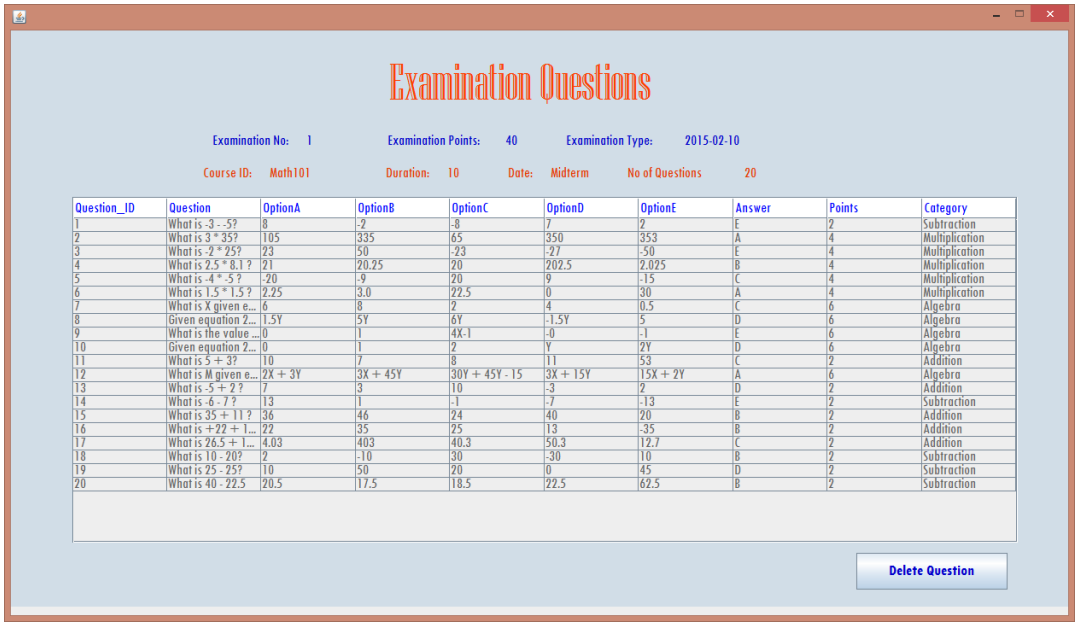


Figure 3.43. Student's grade and performance page

Viewing Exam Questions: Teachers also have the option of viewing and editing the examination questions that they set. Questions can be deleted using the "Delete Questions" button. The page also displays the course name, number of question set, exam duration, exam date, exam points and exam type. Figure 3.44 gives an illustration of the "View examination Questions" page.



Question_ID	Question	OptionA	OptionB	OptionC	OptionD	OptionE	Answer	Points	Category
1	What is -3 - -5?	8	-2	-8	7	2	E	2	Subtraction
2	What is 3 * 35?	105	335	65	350	353	A	4	Multiplication
3	What is -2 * 25?	23	50	-23	-27	-50	E	4	Multiplication
4	What is 2.5 * 8.1 ?	21	20.25	20	202.5	2.025	B	4	Multiplication
5	What is -4 * -5 ?	-20	-9	20	9	-15	C	4	Multiplication
6	What is 1.5 * 1.5 ?	2.25	3.0	22.5	0	30	A	4	Multiplication
7	What is X given e...	6	8	2	4	0.5	C	6	Algebra
8	Given equation 2...	1.5Y	5Y	6Y	-1.5Y	5	D	6	Algebra
9	What is the value ...	0	1	4X-1	-0	-1	E	6	Algebra
10	Given equation 2...	0	1	2	Y	2Y	D	6	Algebra
11	What is 5 + 3?	10	7	8	11	53	C	2	Addition
12	What is M given e...	2X + 3Y	3X + 45Y	30Y + 45Y - 15	3X + 15Y	15X + 2Y	A	6	Algebra
13	What is -5 + 2 ?	7	3	10	-3	2	D	2	Addition
14	What is -6 - 7 ?	13	1	-1	-7	-13	E	2	Subtraction
15	What is 35 + 11 ?	36	46	24	40	20	B	2	Addition
16	What is +22 + 1...	22	35	25	13	-35	B	2	Addition
17	What is 26.5 + 1....	4.03	403	40.3	50.3	12.7	C	2	Addition
18	What is 10 - 20?	2	-10	30	-30	10	B	2	Subtraction
19	What is 25 - 25?	10	50	20	0	45	D	2	Subtraction
20	What is 40 - 22.5	20.5	17.5	18.5	22.5	62.5	B	2	Subtraction

Figure 3.44. Examination questions

Viewing Student Total and Performance: Teachers can view student's total and average performance by selecting "View Student's total and Performance" from the start page dropdown options. The teacher assistant agent and the Admin Info Extract agent help in providing this information. Figure 3.45 shows the student's total and performance page. The average performance is displayed at the bottom of the page.

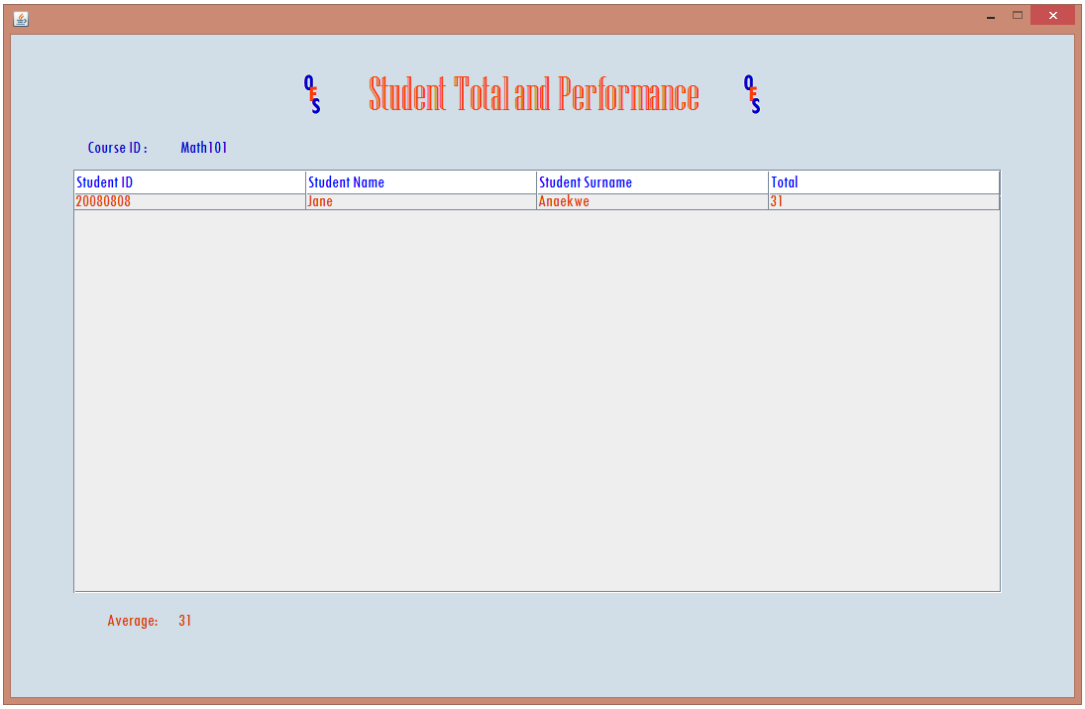


Figure 3.45. Student total and average performance

Viewing Exam Attendance: The system automatically keeps track of students who attended the exam, eliminating the need for manual collection of exam attendance signature from students. With this teachers can know which student attended the exam and how many they are. Figure 3.46 gives a description of the exam attendance page. The total number of attendees is displayed at the bottom of the page.

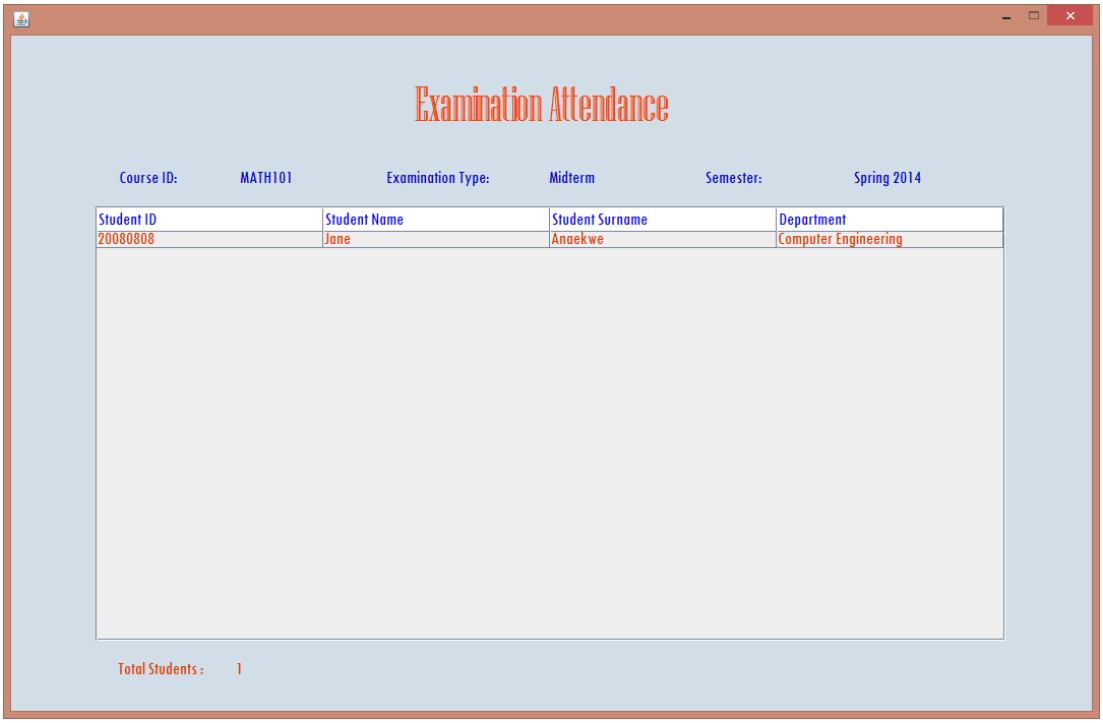


Figure 3.46. Exam Attendance

Viewing Agent Suggestions: Teachers can view suggestions given by agents concerning the areas students need to improve in. By viewing this suggestions teachers, can have an idea about the weak point of students and try to bring up solutions to the problem. In Figure 3.47 the teacher assistant agent shows a math teacher the performance percentage of his student in the various math topics and tells him that his student need to improve in addition and subtraction.

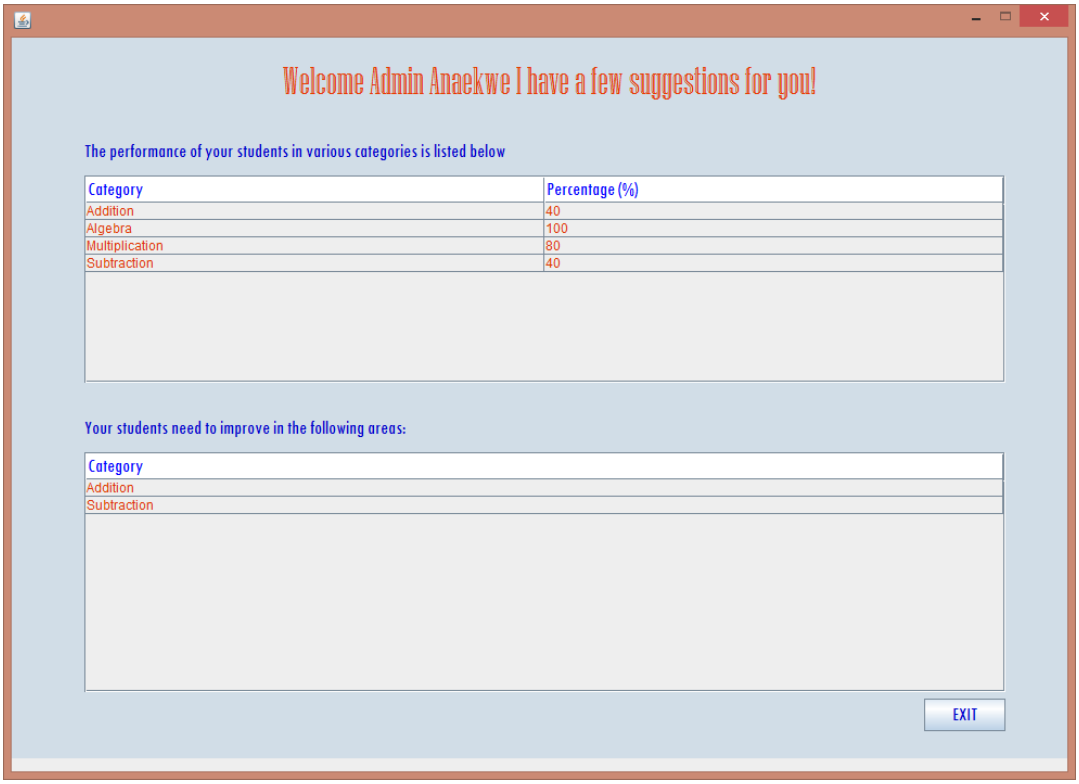


Figure 3.47. Agent suggestions for Math teacher

Viewing Student Exam Requests: Students are given the opportunity to make request during an examination; they can either request for the teachers attention or request for extra time. Once the teacher accesses the system, he/she can view student requests from the student requests page and take necessary actions. The teacher can give extra time to student from the student requests page and the extra time is automatically added to students remaining time by the exam monitor agent. Figure 3.48 shows the student request page.

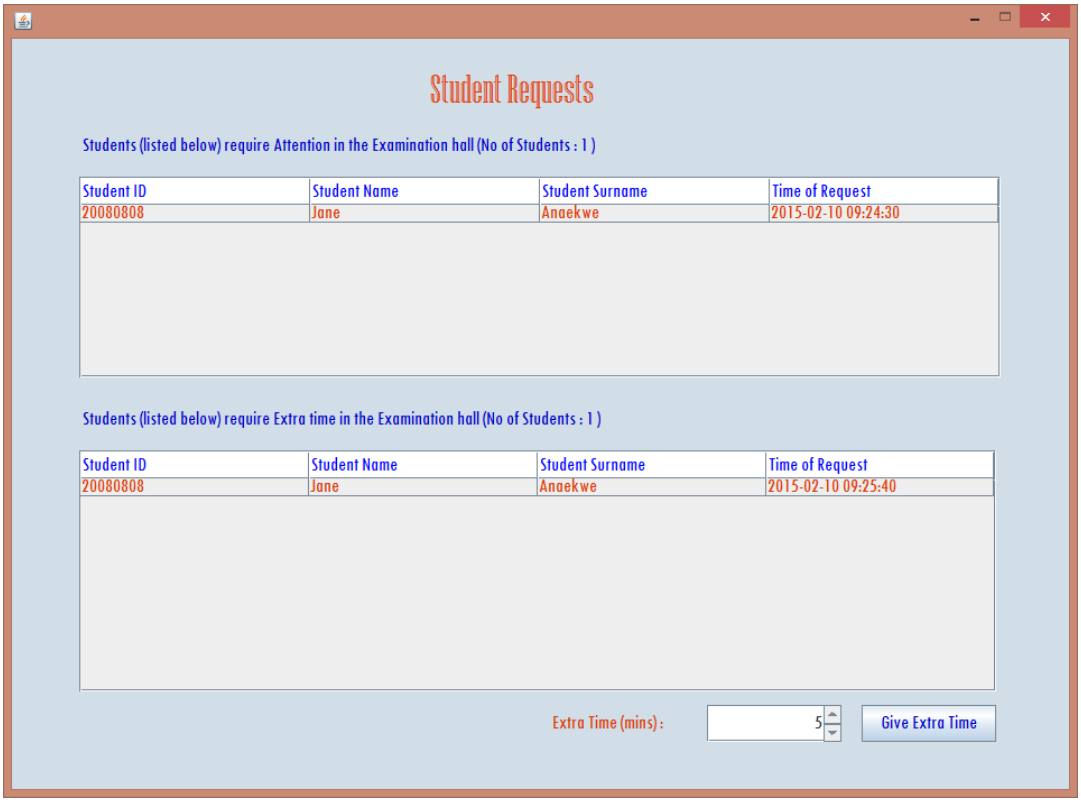


Figure 3.48. Student requests page

Student's Start Page: Upon successful login by students, the student's start page is displayed. Student can view their semester courses and can select operations they want to perform from the "Options" dropdown button. Figure 3.49 shows the student start page.

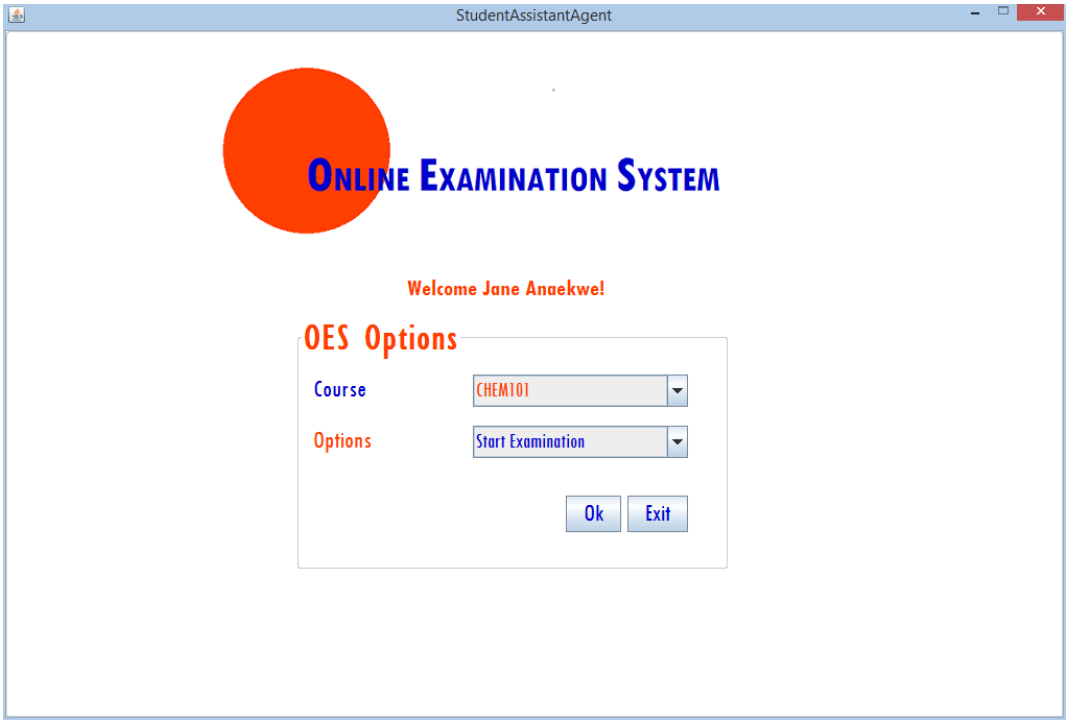


Figure 3.49. Student Start Page

Taking an Exam: As soon as an exam is activated by the exam server agent, students can freely take exams on the system. If the exam is not activated or if the exam has already been taken an "Exam has already been taken or is not available" message will be displayed on the exam page. During an exam students can see the number of answered questions, number of unanswered questions, remaining minutes and estimated time to finish. Students can request for the teacher's attention by clicking on the "Request for teacher" button or they can request for extra time by clicking on the "Request for extra time" button. The "Request for Extra time" button is only enabled at a certain time and can only be clicked once; the button is disabled once it is clicked. Students also have an opportunity to answer questions they skipped by clicking on the "unanswered questions" button after going through all the examination questions. This gives students the opportunity to answer easy questions first and then attempt the

difficult one later; this saves student's time in the exam hall. The "unanswered questions" button is only activated after students have gone through all the exam questions. The exam monitor agents also provides "hurry up" reminders to the students when they are running out of time (as shown in Figure 3.50 on the right bottom corner). Student can also take practice exams in order to improve their skills and get familiar with the system. The outlook of practice exams is similar to that of figure 3.50 except for the request buttons.

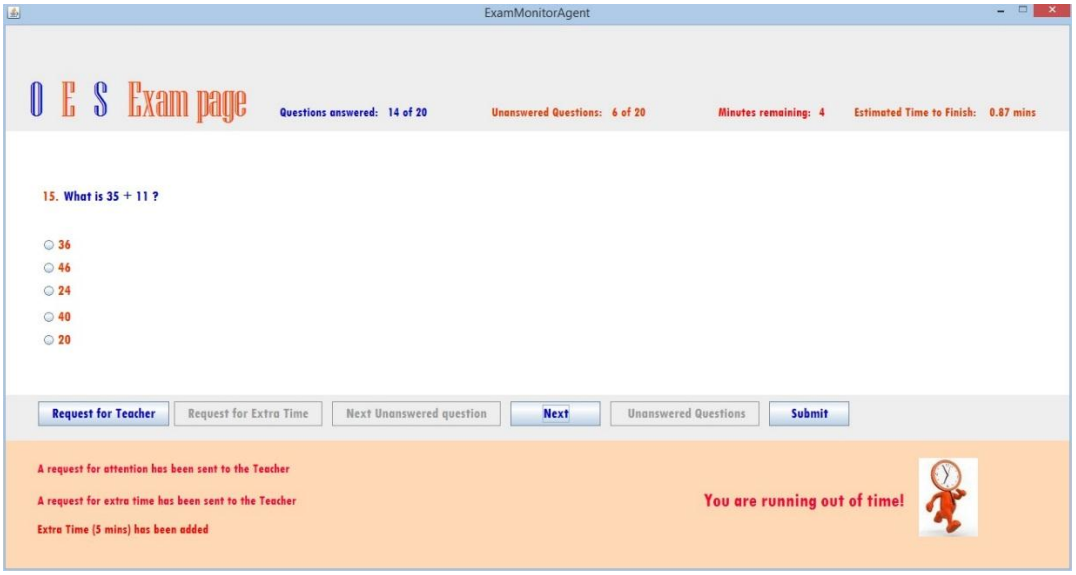


Figure 3.50. Examination Page

Viewing Overall Performance: Student can view student their overall performance by selecting "View Overall Performance" from the start page dropdown options. The points for each exam taken is displayed together with the points gotten by the student. The total of the student is also displayed at the bottom of the page as shown in figure 3.51.

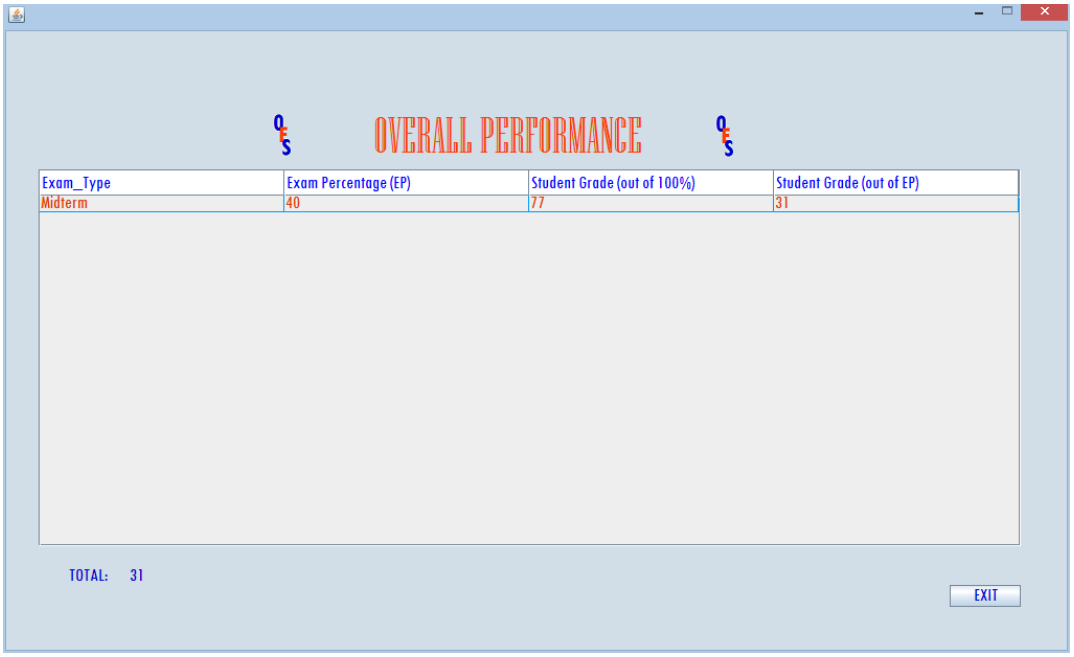


Figure 3.51. Overall Performance

Viewing Performance by Subject: Students can view the grades gotten in the various topics covered in an exam. For example, Figure 3.52 show the Mathematics exam grades gotten by a student in addition, subtraction, algebra and multiplication. In this way the student can discover which topics they performed good or bad in.

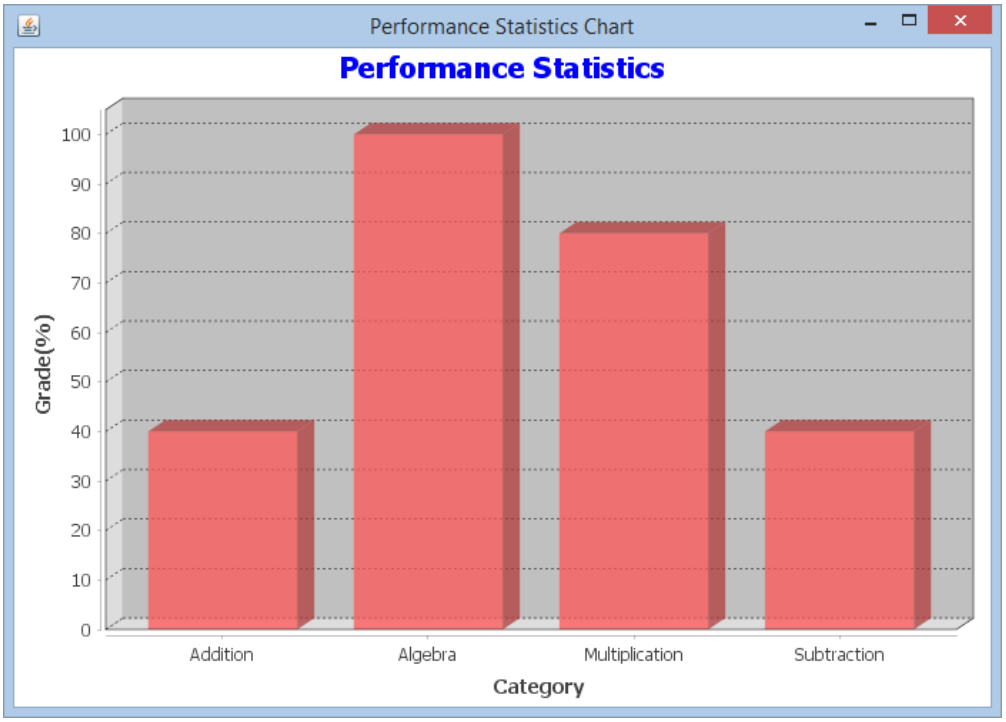


Figure 3.52. Performance by Subject

Viewing Agent suggestion: Students can view suggestions given by agents concerning the areas they students need to improve in. By viewing this suggestions, students can have an idea about their weak points and work towards improving. In Figure 3.53 the student assistant agent shows a student her performance percentage in various math topics and tells her that she needs to improve in addition and algebra.

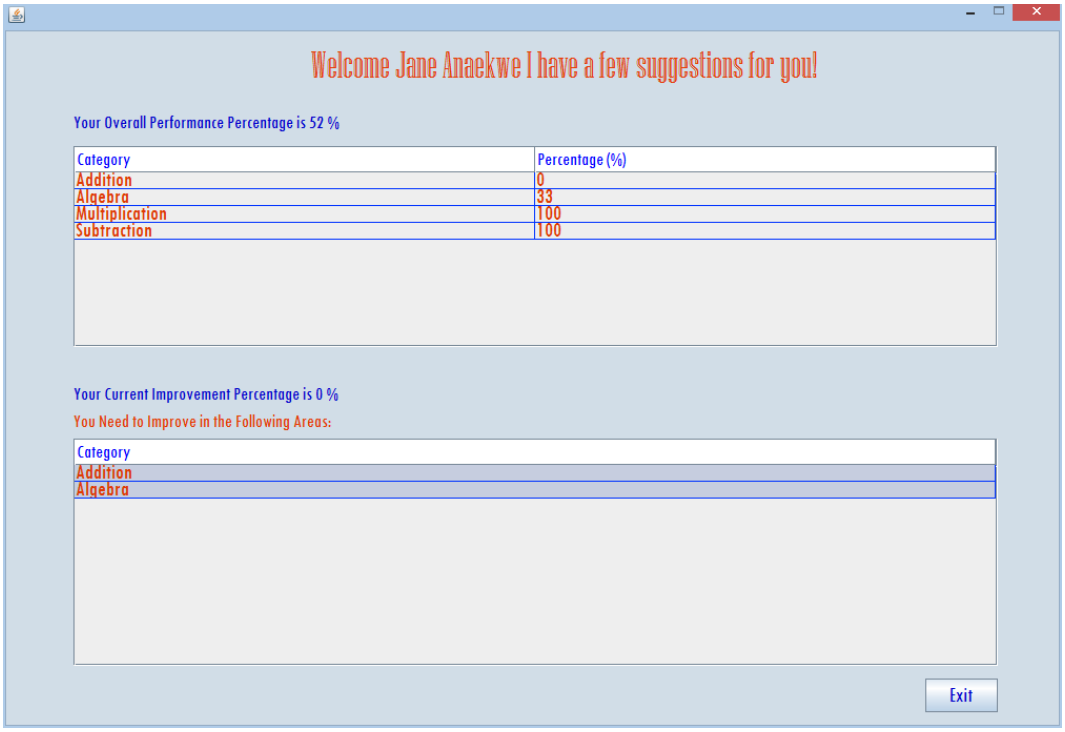
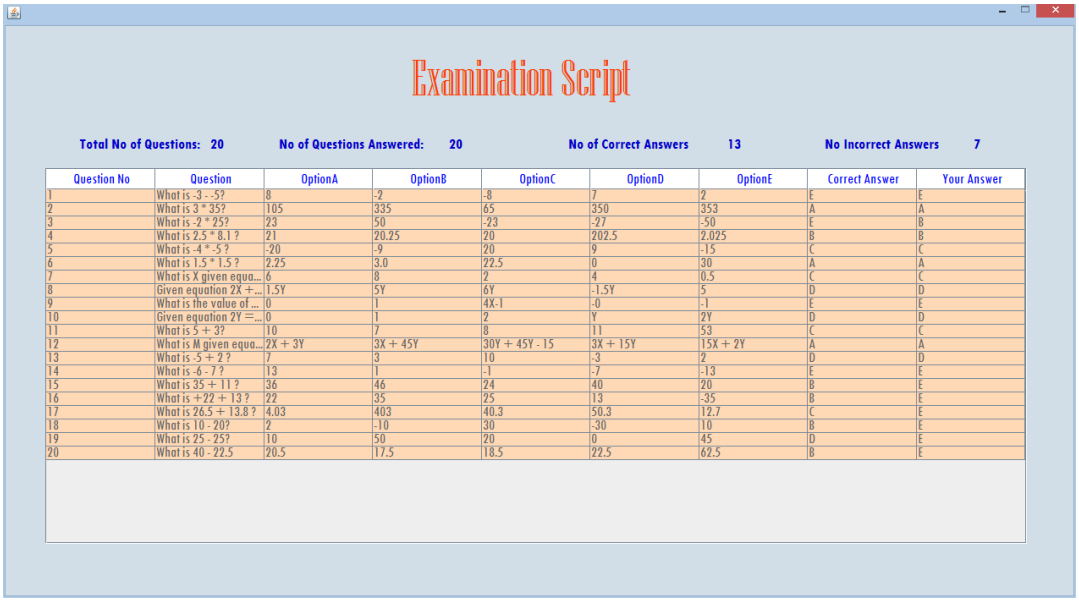


Figure 3.53. Agent suggestions to Student

View Exam Script: Students can view their exam script immediately after an exam. They can view the questions they answered and compare their answers with the correct answers. The total number of questions, number of correct, incorrect answers and the number of questions answered are all displayed to the student on the exam script page by the exam monitor agent.



Question No	Question	OptionA	OptionB	OptionC	OptionD	OptionE	Correct Answer	Your Answer
1	What is -3 - -5?	8	-2	-8	7	2	E	E
2	What is 3 * 35?	105	335	65	350	353	A	A
3	What is -2 * 25?	23	50	-23	-27	-50	E	B
4	What is 2.5 * 8.1 ?	21	20.25	20	202.5	2.025	B	B
5	What is -4 * -5 ?	-20	-9	20	9	-15	C	C
6	What is 1.5 * 1.5 ?	2.25	3.0	22.5	0	30	A	A
7	What is X given equa...	6	8	2	4	0.5	C	C
8	Given equation 2X +...	1.5Y	5Y	6Y	-1.5Y	5	D	D
9	What is the value of ...	0	1	4X-1	-0	-1	E	E
10	Given equation 2Y =...	0	1	2	Y	2Y	D	D
11	What is 5 + 3?	10	7	8	11	53	C	C
12	What is M given equa...	2X + 3Y	3X + 45Y	30Y + 45Y - 15	3X + 15Y	15X + 2Y	A	A
13	What is -5 + 2 ?	7	3	10	-3	2	D	D
14	What is -6 - 7 ?	13	1	-1	-7	-13	E	E
15	What is 35 + 11 ?	36	46	24	40	20	B	E
16	What is +22 + 13 ?	22	35	25	13	-35	B	E
17	What is 26.5 + 13.8 ?	4.03	403	40.3	50.3	12.7	C	E
18	What is 10 - 20?	2	-10	30	-30	10	B	E
19	What is 25 - 25?	10	50	20	0	45	D	E
20	What is 40 - 22.5	20.5	17.5	18.5	22.5	62.5	B	E

Figure 3.54. Examination Script

3.7 **Benefits of the System**

The intelligent online multi-agent system has a lot of advantages. Some of them are:

- It provides a smart efficient way of handling examination as compared to the paper-based examinations discussed in section 1.1.
- It is an online system that provides flexibility and convenience to students and teachers, e.g. students can request for the teacher's attention just by the click of a button. Students can request for extra time in the exam halls and if given by the teacher time information is automatically updated on the system. Teachers can handle this extra time requests without being present in the exam hall.
- It reduces the workload that comes with handling exams e.g. distribution and grading of large amount of papers, taking exam attendance manually which causes delays and distractions in examples.

- It eliminates delays in declaration of exam results and creates an atmosphere of fairness.
- The agents introduce intelligence into the system by providing fun and helpful features such as the hurry-up reminders when time is running out, extra time or attention requests, estimated time to finish calculations, agent suggestions and performance analysis. The suggestions given by agents help students and teachers to easily analyse the weak points of students and address them. These features, help in making the system a conducive environment for knowledge evaluation.
- Practice exams provided help students improve their knowledge and learning skills.
- Functions of the system is distributed over a multi-agent framework which make it easier to handle.
- It improves the quality of teaching and learning by providing services that satisfy the needs of both teachers and students.

Chapter 4

THE IMPLEMENTATION PROCESS

4.1 The JADE Platform

The Jade Platform [14] together with the Net beans IDE was used in implementing this thesis. In this chapter the implementation process of the thesis will be discussed in detail.

4.1.1 What is JADE?

JADE (Java Agent Development Framework) is a middleware or software framework used for developing multi-agents systems in accordance with FIPA (Foundation for İntelligent Physical Agents) specifications [20], [35], [36]. JADE applications have peer-to-peer architecture that enable agents to interact and interoperate concurrently [20].

4.2 How are JADE Agents Created?

Agents as described in section 2.1 are software components or programs that are capable of performing function autonomously on behalf of users. JADE agents are basically Java programs that are simply created by defining java classes with the jade.core.Agent extension [35]. A code sample of how Agents are created is shown in figure 4.1.

```
import jade.core.Agent;
public class LoginAgent extends Agent {
   protected void setup() {
public Behaviour send = new Behaviour() {
                System.out.println("Agent created....");
          public void action() {
       System.out.println("Agent working....");  }}
   protected void takeDown() {
       System.out.println("Agent terminating....") }}}
```

Figure 4.1. A JADE agent

The setup method shown in figure 4.1 is used for specifying agent initializations when an agent is created [35], [14]. The takedown method is used for performing clean-up operations just before an agent is terminated [14]. The behaviour of agents are implemented in the action method. The action method is used for defining what operations should be performed when the behaviour of an agent is executing [14], [35].

In order for the JADE agents to be executed the JADE runtime environment must be started using the Jade.Boot command. JADE runtime environments are known as *containers* in which JADE agents live [35]. A collection of one or more containers is known as a *platform* [37]. Agents in one container or platform can communicate with agents in other containers or platforms in a network [37].

Every JADE container has two default agents: AMS (Agent Management System) and DF (Directory Facilitator) [35], [37]. The AMS agent performs Platform management

operations such as shutting down agent platforms or starting and killing agents [37]. The DF agent provides a yellow pages service through which agents can publish the services they provide, and also find other agents that provide the services they need [35], [37]. Figure 4.2 show the general architecture of a JADE multi-agent system. The symbols A1-A5 represent agents living in containers.

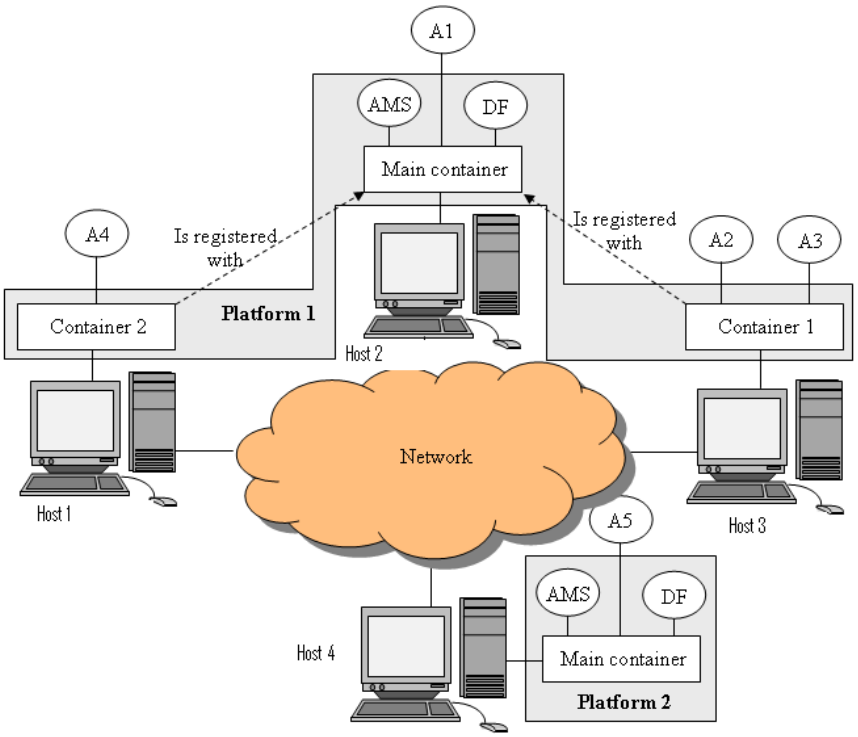


Figure 4.2. General JADE Architecture [37]

Agents in a container can be managed by developers using the JADE Management Console [37]. Figure 4.3 shows the JADE management console for the agents on the server-side of the multi-agent OES system.

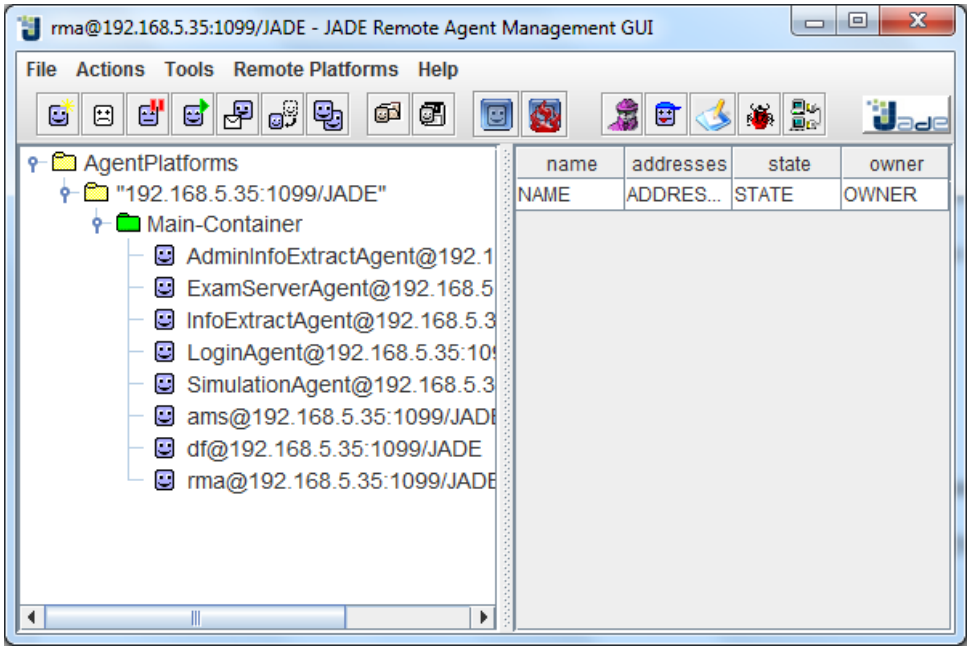


Figure 4.3. JADE management console for server agents

4.3 How do JADE Agents communicate?

JADE agents communicate with the use of ACL (Agent Communication language) languages defined by FIPA [14], [20], [35], [37]. An ACL message consist of a number of fields such as:

- The message ***sender*** [37]
- The ***receivers*** [37]
- The ***performative*** or ***communicative*** act: This describes the intention of the sender. The communicative intention could be a REQUEST, INFORM, PROPOSE, or CFP (Call for proposal) [14], [35], [37].
- The ***content*****:** The actual information contain in a message. For example, a INFORM message could contain information about the grade of a student [14], [35], [37].

- The ***content language:*** This is the language or syntax used to express the message content. This language must be understandable to both the sending and receiving agent in order to communicate effectively [14], [35].
- The ***ontology*****:** Used in indicating the vocabulary of characters or symbols used in a message. For effective communication, the meaning of these characters and symbols must be the same for communicating agents [14], [35].

The figure 4.4 below shows a code snippet of how the Login user agent sends a request message to the login agent on the server. Messages are sent by agents using the send method [14]. As shown in figure 4.4. Other examples of agent communication can be found in the appendix.

```
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("LoginAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("Login");
request.setContent(LoginGui.username);
send(request);
```

Figure 4.4. Agent Communication: Sending messages

Messages are received by JADE agents using the receive or blockingReceive method [35]. Figure 4.5 shows a simple sample code of how agents receive messages.

```
ACLMessage msg = myAgent.receive(template);
        if (msg != null) {
            message = msg.getContent();
        }
```

Figure 4.5. Agent Communication: Receiving messages

4.4 **How do JADE agents behave?**

The functions or job of an agent is described by its behaviour [35]. JADE has many predefined behaviours e.g. One-shot behaviour (actions are executed only once), Cyclic behaviour (actions are performed repetitively) [35]. JADE agent behaviours are implemented as objects of agent classes that extends jade.core.behaviours.Behaviour [14], [35]. The action method of an agent behaviour can be programmed according to the desires of the developer. Figure 4.6 shows the behaviour of an agent whose job is to show a hello message just once. Other example of agent behaviours can be found in the appendix.

```
Behaviour openAcess = new OneShotBehaviour() {
        public void action() {
            System.out.println("Hello I'm an agent);
            }
```

Figure 4.6. JADE Agent behaviour

4.5 **JADE Agent Internal Structure**

Once an agent is created and executed, it begins by performing the initial operations in it's setup method. After initialization and initial actions are completed, it begins performing the behaviour actions specified in its agent class. One behaviour is performed after another; the done method is used to check if the behaviour has been completed or not. Once completed, it is removed from the pool of agent behaviour. When an agent is terminated, the takedown method is called to execute clean-up operations before termination.

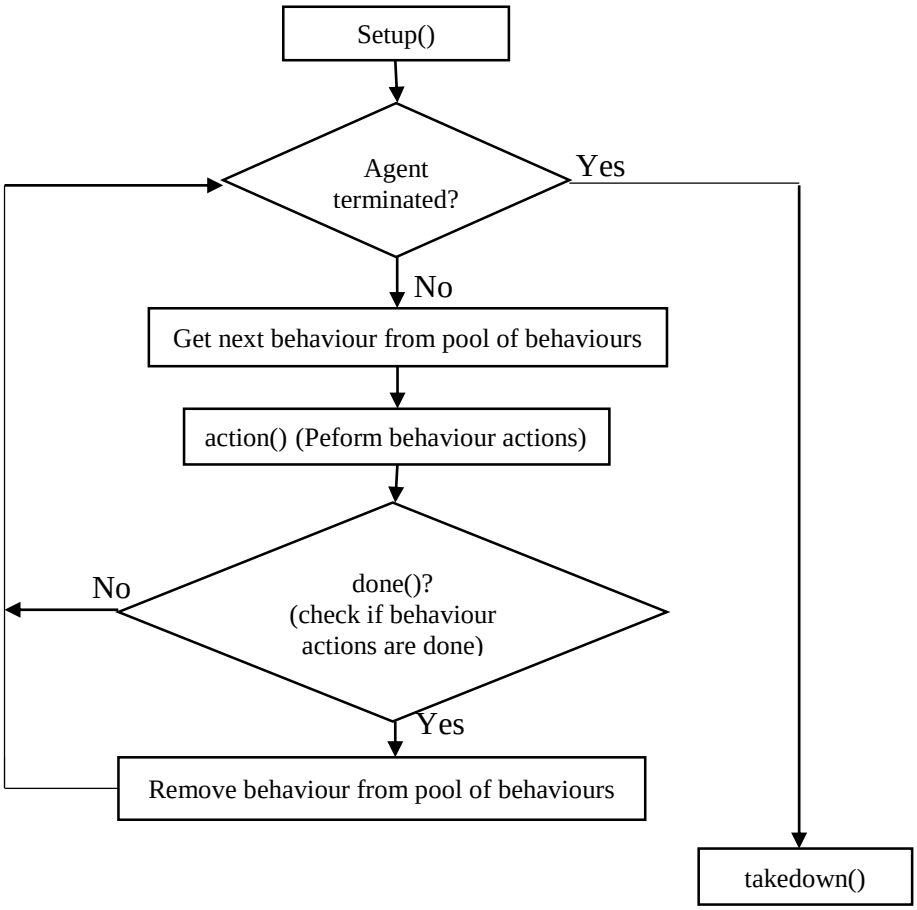


Figure 4.7. JADE agent internal structure [14]

4.6 System Implementation Design

In this section a general descriptive illustration of how the system was designed will be shown. For deeper understanding of how the system works the implementation and design of some parts of the system will be described. As mentioned before in section 3.5, there are five agents on the user ansd server side of the system. The agents on the server are constantly running and waiting for requests from user agents or for changes in their environment (the OES system). On the other hand the agents on the server side only start running when the OES system is started by a user.

4.6.1 Login Phase

Once the OES system is started by a student or teacher and the user agents are properly running on the user PC (as depicted in figure 3.29, section 3.5), the activities on the system are now ready to be performed. The first action performed when the system is started is the login operation. Once a user enters his/her username and password and clicks the login button, the login user agent immediately sends an authentication request to the login agent on the server side takes with the information entered by the user. The server login agent receives the information sent by login user agent, queries the system database (the student table, and admistration table, section 3.4) for information about users and compares the results from the database with the information sent by the login user agent. If the database information matches the data sent, a message successfully validating the user is sent back to the login user agent. If there is no match a 'invalid' message is sent back and system access is denied. Figure 4.8 shows the login phase system design.

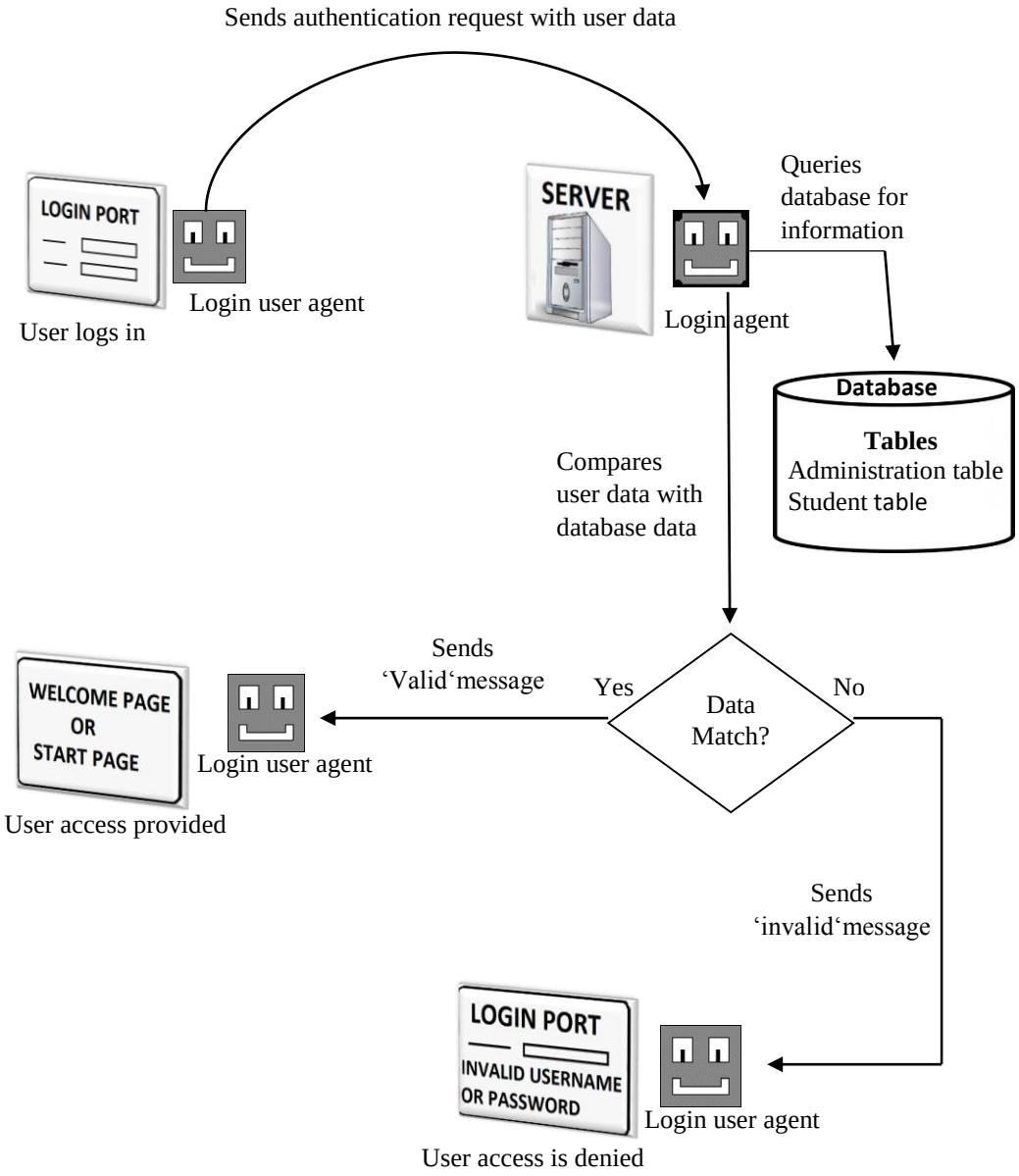


Figure 4.8. Login phase system design

```
public Behaviour send = new Behaviour() {
public void action() {// actions of the login user agent is defined below
System.out.println("Agent " + getLocalName() + ": sending REQUEST  message...");
//creating a request message
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
//creating message receiver
AID aid = new AID("LoginAgent@192.168.5.35:1099/JADE", AID.ISGUID);
//setting receiver address
aid.addAddresses("http://192.168.5.35:7778/acc");+
//adding receiver
request.addReceiver(aid);
//setting the message ontology
request.setOntology("Login");
//Setting message content information entered by user
request.setContent(LoginGui.username + "~" + LoginGui.Pswd + "~" +
LoginGui.userType);
//sending message
send(request);
System.out.println("Agent " + getLocalName() + ": sent REQUEST message..." +
LoginGui.username + "-" + LoginGui.Pswd + "-" + LoginGui.userType);
//adding behaviour for recieving message response
myAgent.addBehaviour(receive);
}
```

Figure 4.9. Code for sending authentication request to the server login agent

The figure 4.9 shows a code snippet of how the login agent on the user-side send an authentication request to the server login agent. The behaviour object ‘send’ defines how the login user agent behaves. The actions of the agent is defined within the action methods and contains all the operations done by the agent’s send behaviour.

```
addBehaviour(new CyclicBehaviour(this) {
public void action() {// actions of the login user agent is defined below
ACLMessage msg = myAgent.receive(template); // receives message
if (msg != null) {
String message = msg.getContent();// saves message content in message string
System.out.println("Agent " + getLocalName() + " recieved REQUEST message");
messageArray = new String[3];
messageArray = message.split("~");// splits message and saves it in a message array
try {// creating db connection
Class.forName("com.mysql.jdbc.Driver");
Connection con = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/onlineexamination",
"root", "ghsm4/24/7/*");
Statement stmt = con.createStatement();
if (messageArray[2].equals("Student")) {// student authentication
// Query for getting student information from database
String sql = "Select student_id,student_name,"
+ "student_surname,student_password "
+ "from student where student_id='" + messageArray[0]
+ "'and student_password='" + messageArray[1] + "'";
// executing sql query
ResultSet rs = stmt.executeQuery(sql);
if (rs.first()) {// Extracting the Query results
String sid = rs.getString("student_id");
String spswd = rs.getString("student_password");
String sn = rs.getString("student_name");
String ssn = rs.getString("student_surname");
if (sid.equals(messageArray[0]) && spswd.equals(messageArray[1])) {
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());// setting message receiver
//setting the message ontology
reply.setOntology("Login");
//Setting 'valid user' message content
reply.setContent("Valid~" + sid + "~" + sn + "~" + ssn);
send(reply);//sending valid message
}} else {ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());
reply.setOntology("Login");
//Setting 'invalid user' message content
reply.setContent("Invalid~Invalid~Invalid~Invalid");
System.out.println("Agent " + getLocalName() + " sending INFORM message");
send(reply);//sending invalid message}}
```

Database query for authenticating students

Extracting database information

Figure 4.10. Code for the receiving/sending messages for student authentication

```
if (messageArray[2].equals("Admin")) {
// Query for getting teacher information from database
String sql = "Select
Admin_username,Admin_name,Admin_password,Admin_surname "
+ "from administration where Admin_username='" + messageArray[0]
+ "'and Admin_password='" + messageArray[1] + "'";
ResultSet rs = stmt.executeQuery(sql);
if (rs.first()) {// Extracting the Query results
String ausid = rs.getString("Admin_username");
String spswd = rs.getString("Admin_password");
String aname = rs.getString("Admin_name");
String asname = rs.getString("Admin_surname");
if (ausid.equals(messageArray[0]) && spswd.equals(messageArray[1])) {
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());
reply.setOntology("Login");
//Setting 'valid user' message content
reply.setContent("Valid~" + ausid + "~" + aname + "~" + asname);
System.out.println("Agent " + getLocalName() + " sending INFORM message"
+ "Valid~" + ausid + "~" + aname + "~" + asname);
send(reply);//sending valid message}
} else {ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());
reply.setOntology("Login");
//Setting 'invalid user' message content
reply.setContent("Invalid~Invalid~Invalid~Invalid");
System.out.println("Agent " + getLocalName() + " sending INFORM message");
send(reply);//sending invalid message}}} catch (Exception e)
{JOptionPane.showMessageDialog(null, e.getMessage());}
} else {block();}}});
```

Database Query and Data extraction

Figure 4.11. Code for the receiving/sending messages for teacher authentication

4.6.2 **Starting an examination**

One of the major activities done on the OES system is taking an exam. There are three agents involved in the exam process; Exam monitor agent, Exam server agent and Info extract agent. When a student is successfully authenticated, the student start up page is displayed. If the student selects the 'Start Examination' option for a particular course, the exam monitor agent immediately sents a request for exam information (exam questions, options, duration, deactivation time) to the Exam server agent. The Exam server agent first checks if the examination is activated or deactivated (i.e. if the time for the examination is due or has elapsed) and it also checks if the exam has already been taken. This is carried out by querying the system database (examination, active_examination, inactive_examination, exam _question_answer, and selected_log table section 3.4). If the examination is activated the necessary exam information is sent to the exam monitor agent and student can begin the exam (depicted in section 3.6). If the exam is not activated or has already been taken an 'exam unavailable or exam already taken' message is sent to the Exam monitor agent and the message is displayed to the user

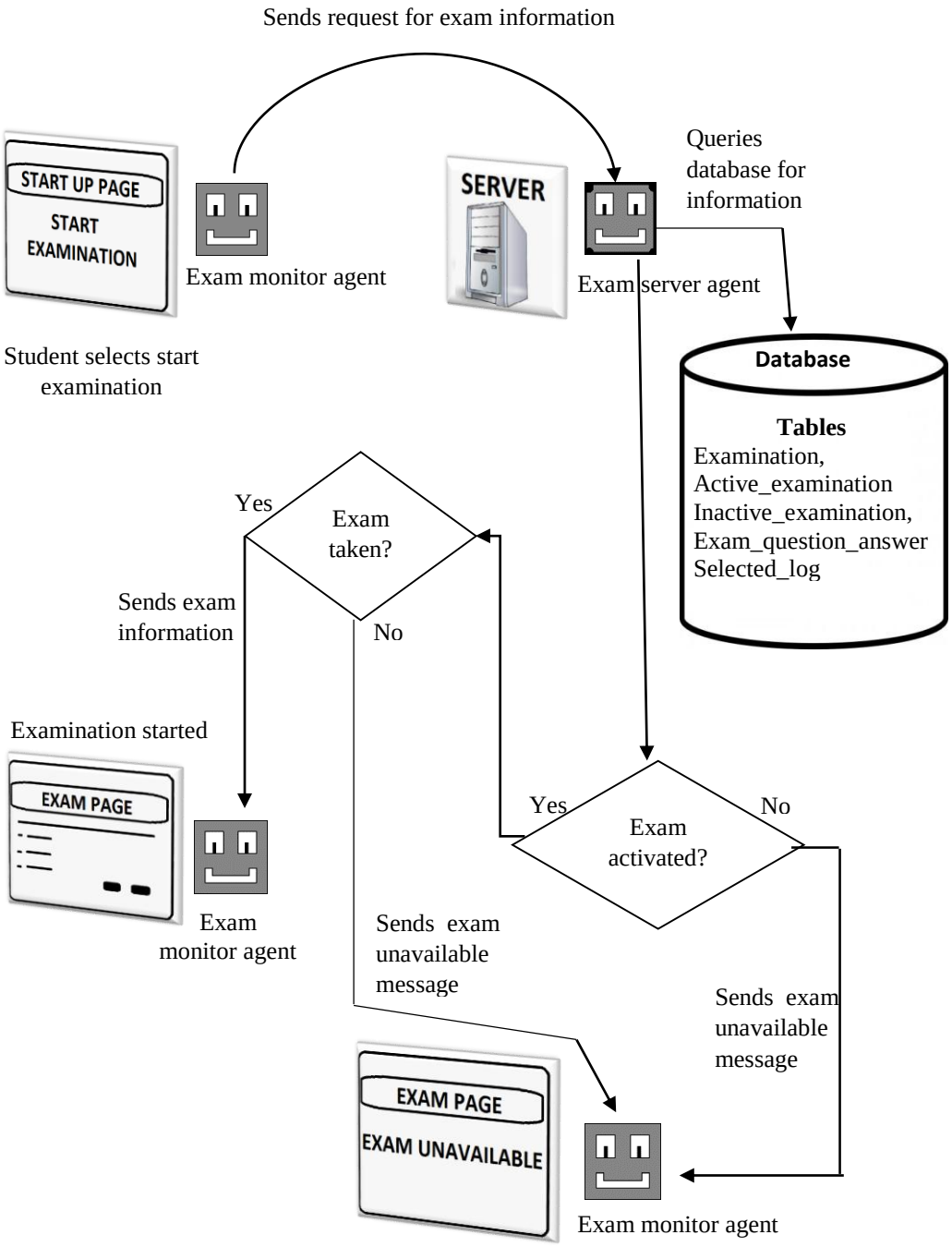


Figure 4.12. Start examination system design

```
//Creating request message
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
//Creating message receiver (ExamServerAgent)
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE",
AID.ISGUID);
//Setting address of receiver
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("GetExamQuestions");
Question_number++;
ExamPage.questionNoLabel.setVisible(true);
ExamPage.questionNoLabel.setText(Question_number + ".");
// setting message content
request.setContent(LoginUserAgent.UserId + "~" + StartPage.Course + "~" +
Question_number);
//sending message request
send(request);
```

Figure 4.13. Code for sending request for exam information

```
ACLMessage requestMessage2 = myAgent.receive(requestTemplate2); //recieving message sent
if (requestMessage2 != null) {
System.out.println("Agent " + getLocalName() + " recieved REQUEST message ");
String message = requestMessage2.getContent();
messageArray2 = new String[3];
messageArray2 = message.split("~");// splitting message into different parts
try {Class.forName("com.mysql.jdbc.Driver");
//SQL query for extracting informatıon
String sql = " SELECT exam_question_answer.Examination_no,Question_ID,Duration,
Question,category "+ " FROM examination,exam_question_answer "
+ " where examination.Examination_No = exam_question_answer.Examination_No "
+ " and exam_question_answer.question_id not in (select Question_ID from selected_log where
student_id='" + messageArray2[0] + "') "+ " and examination.Examination_No in (select
Examination_No from active_examination)"
+ " and examination.Examination_No not in (select Examination_No from
inactive_examination)"+ " and examination.Examination_No not in (select Examination_No
from exams_taken where Student_ID= '" + messageArray2[0] + "') "
+ " and Course_ID= '" + messageArray2[1] + "'";
Connection con = (Connection) //creating SQL connection
DriverManager.getConnection("jdbc:mysql://localhost:3306/onlineexamination",
"root", "ghsm4/24/7/*"); Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(sql);
if (rs.first()) {// extracting data and creating reply message
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(requestMessage2.getSender());
reply.setOntology("ExamQuestions");
String ENo = rs.getString("Examination_No");
String Qid = rs.getString("Question_ID");
String Question = rs.getString("Question");
String Category = rs.getString("Category");
reply.setContent("Valid~" + ENo + "~" + Qid + "~" + Question + "~" + Category);// setting
message content with information extracted
System.out.println("Agent " + getLocalName() + " sending INFORM message Valid-" + ENo +
"-" + Qid + "-" + Question + "-" + Category);
send(reply);// sending response if available
sql = "insert into selected_log (Examination_no,Question_ID,Question_No,Student_ID) "+ "
values ( '" + ENo + "','" + Qid + "'," + messageArray2[2] + ",'" + messageArray2[0] + "')";// SQL
for inserting questions sent to student
stmt.executeUpdate(sql);
} else {ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(requestMessage2.getSender());
reply.setOntology("ExamQuestions");
reply.setContent("Invalid~Invalid~Invalid~Invalid~Invalid");
System.out.println("Agent " + getLocalName() + " sending INFORM message Valid");
send(reply);// sending invalid message when questions are unavailable}
} catch (Exception e) {JOptionPane.showMessageDialog(null, e.getMessage());}} else{block();}
```

Query execution and data extraction

Figure 4.14. Code for receiving/responding to exam information request

4.6.3 **Handling student exam requests**

As mentioned previously in section 3.5.2 and 3.6, students can make request during an exam. They can either request for the teacher's attention in the exam hall by clicking the 'Request for teacher button' or they can request for extra time by clicking the 'request for extra time' button. If the student clicks the 'Request for teacher' button the Exam monitor agent immediately send the request to the info extract agent on the server. The Info extract agent inserts this request into the database (Student_requests table, section 3.4) and responds by sending a 'request was successfully sent' message which is displayed to the student by the Exam monitor agent.

```
System.out.println("Agent " + getLocalName() + ": sending REQUEST message...");
//Creating request message
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
//Creating message receiver (InfoExtractAgent)
AID aid = new AID("InfoExtractAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");//Setting address of receiver
request.addReceiver(aid);
request.setOntology("InsertStudentRequest");
request.setContent(LoginUserAgent.UserId + "~" + LoginUserAgent.name_of_user +
"~" + LoginUserAgent.surname_of_user + "~" + StartPage.Course
+ "~" + ExamPage.requestType + "~" + Exam_type); // setting message content
send(request); //sending message request
myAgent.addBehaviour(recieveStudentRequestResponse);
```

Figure 4.15. Code for sending exam requests

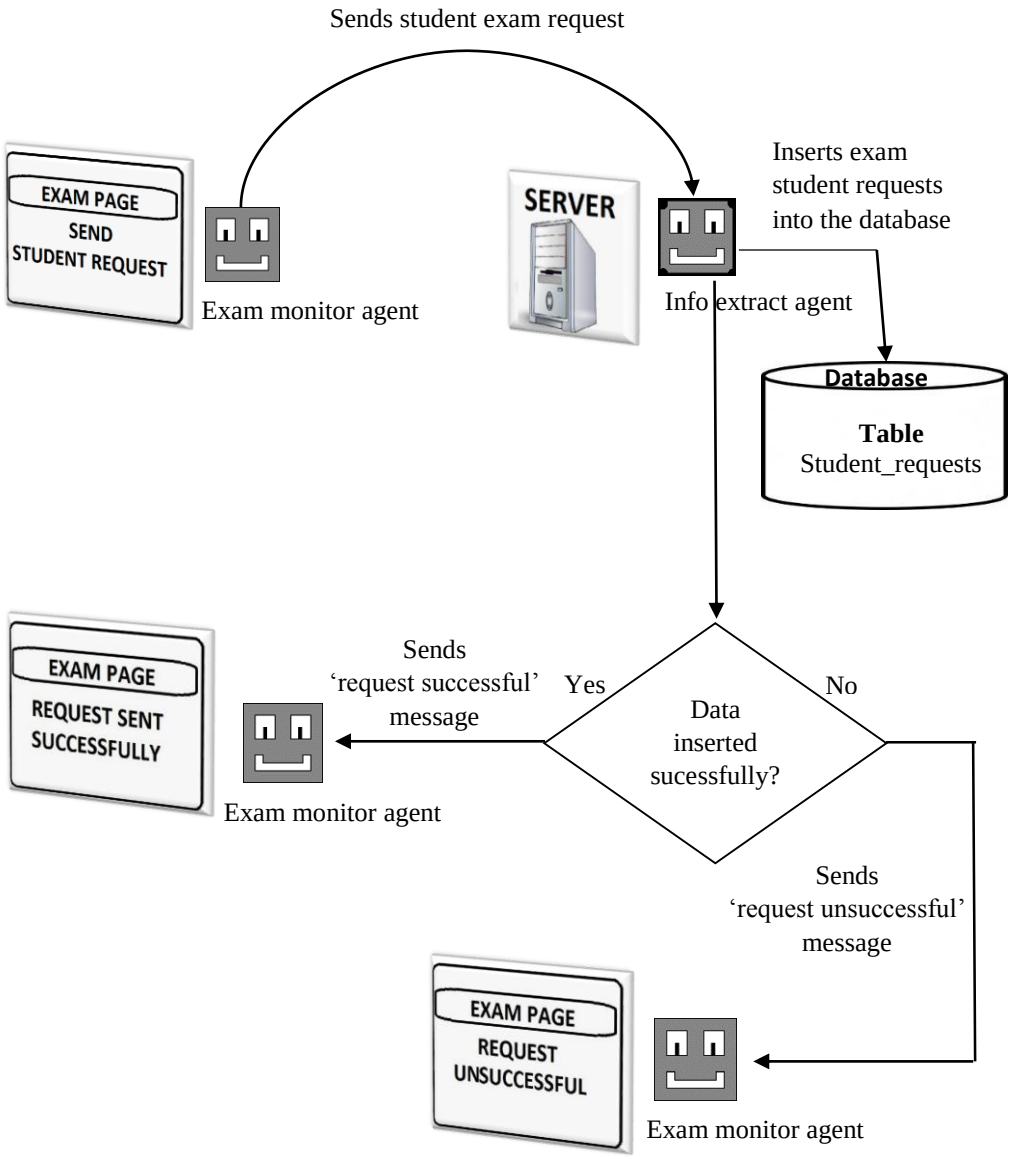


Figure 4.16. Student exam request design

```
if (msg.getOntology().equals("InsertStudentRequest")) {
String message = msg.getContent();// getting message content
messageArray = new String[6];
messageArray = message.split("~");// splitting message into parts
System.out.println("Agent " + getLocalName() + " recieved REQUEST message " +
message);
try {Class.forName("com.mysql.jdbc.Driver");
Connection con = (Connection) //creating SQL connection
DriverManager.getConnection("jdbc:mysql://localhost:3306/onlineexamination",
"root", "ghsm4/24/7/*");
Statement stmt = con.createStatement();
DateFormat dtform = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
Date dtobj = new Date();
//SQL query for extracting information
String sql = "insert into student_requests values('" + messageArray[0] + "','" +
messageArray[1] + "','" + messageArray[2] + "','" + messageArray[3] + "','" +
messageArray[4] + "','" + dtform.format(dtobj) + "','" + messageArray[5] + "');";
stmt.executeUpdate(sql);
if (messageArray[4].equals("Attention")) {
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());// adding receiver
reply.setOntology("StudentRequestResponse");
reply.setContent("Attention~A request for attention has been sent to the Teacher");
System.out.println("Agent " + getLocalName() + " sending INFORM message " +
reply.getContent());
send(reply);}// sending response for attention request
if (messageArray[4].equals("Extra time")) {
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);// creating rply
reply.addReceiver(msg.getSender());
reply.setOntology("StudentRequestResponse");
reply.setContent("Extra time~A request for extra time has been sent to the Teacher");
System.out.println("Agent " + getLocalName() + " sending INFORM message " +
reply.getContent());
send(reply);}} // sending response for extra time  request
catch (Exception e) {JOptionPane.showMessageDialog(null, e.getMessage());
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());
reply.setOntology("StudentRequestResponse");
reply.setContent("Error: Could not send Request there seems to be a Problem");
System.out.println("Agent " + getLocalName() + " sending INFORM message " +
reply.getContent());
send(reply);}}// sending response if failure occurs
```

Figure 4.17. Code for receiving/sending request response message

When a teacher select the ‘view students request’ option from the teacher start up page, the Teacher assistant agent sends a message to the Admin info extract agent on the server requesting for information about student exam request page. The Admin info extract agent responds by quering the database (Student_requests table, section 3.4). If any request is available the Admin info sends it to the teacher assistant agent who displays this information on the ‘view student request’ page (depicted in section 3.6).

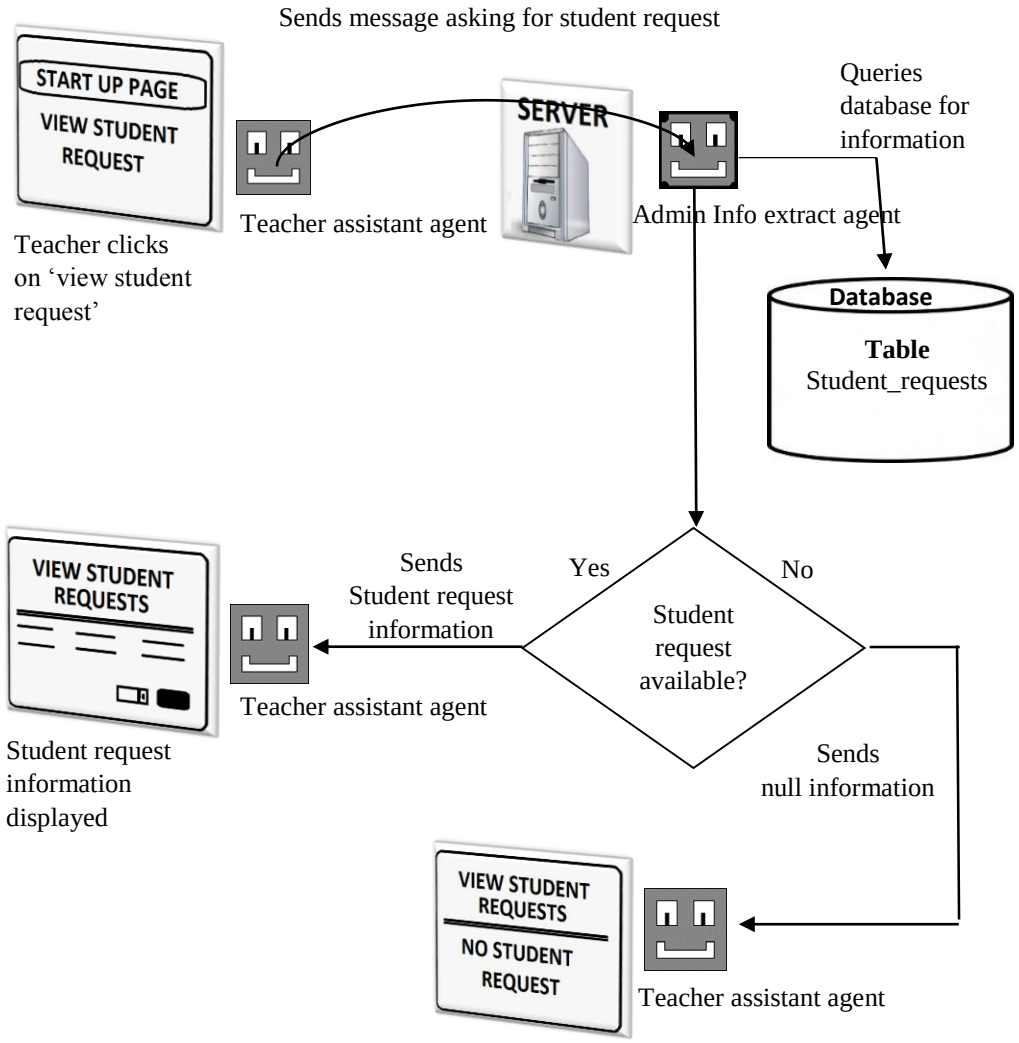


Figure 4.18. Viewing student request design

```
//behaviour for getting student request
public Behaviour getStudentRequests = new OneShotBehaviour() {
public void action() {
System.out.println("Agent " + getLocalName() + ": sending STUDENT
REQUEST message...");
// creating request message
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("AdminInfoExtractAgent@192.168.5.35:1099/JADE",
AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");// setting receiver address
request.addReceiver(aid);//adding receiver
request.setOntology("GetStudentRequests"); // setting ontology
request.setContent(TeacherOptions.Course + "~" + TeacherOptions.eType);
send(request);// sending request message
System.out.println("Agent " + getLocalName() + ": sent REQUEST
message...");}};
```

Figure 4.19. Code for requesting for student exam request

```
if (msg.getOntology().equals("GetStudentRequests")) {String message = msg.getContent();
messageArray = new String[2];nmessageArray = message.split("~");
System.out.println("Agent " + getLocalName() + " recieved REQUEST message " +
message);
String sql = "select Request,count(distinct(student_ID)) as cnt\n"
+ " from student_requests\n"// sql quesry for getting number of requests
+ " where Course_ID= '" + messageArray[0] + "'\n"
+ " and ExamType='" + messageArray[1] + "'\n"
+ " group by Request;";
try {Class.forName("com.mysql.jdbc.Driver"); Connection con = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/onlineexamination",
"root", "ghsm4/24/7/*");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(sql);
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);// creating message
reply.addReceiver(msg.getSender());
reply.setOntology("StudentRequests");
String requestTypes = "";int itemsLength = 0;
while (rs.next()) {String request = rs.getString("Request"); int cnt = rs.getInt("cnt");
requestTypes = requestTypes + "~" + request + "~ Students (listed below) require " + request
+ " in the Examination hall (No of Students : " + cnt + " )";
itemsLength = itemsLength + 2;}
int itemCheck = itemsLength;
sql = "select Student_ID,Student_Name,Student_Surname,Request,Time_of_request\n"
+ " from student_requests\n"// SQL query getting student requests
+ " where Course_ID= '" + messageArray[0] + "'\n"
+ " and ExamType='" + messageArray[1] + "';";
rs = stmt.executeQuery(sql);
while (rs.next()) {String sid = rs.getString("Student_ID");
String stdname = rs.getString("Student_Name");
String stdsname = rs.getString("Student_Surname");
String request = rs.getString("Request");
Date Time_or_request = rs.getDate("Time_of_request");
Time tm = rs.getTime("Time_of_request");
requestTypes = requestTypes + "~" + sid + "~" + stdname + "~" + stdsname + "~" + request +
"~" + Time_or_request + " " + tm;
itemsLength = itemsLength + 5;}
if (requestTypes.equals("")) {itemsLength++;
requestTypes = itemsLength + "~0~0~0~0~0~0~0~0~0~0";} else {
requestTypes = itemCheck + "" + requestTypes; itemsLength = itemsLength++;
requestTypes = itemsLength + "~" + requestTypes;}
reply.setContent(requestTypes);// setting message content
System.out.println("Agent " + getLocalName() + " sending INFORM message " +
requestTypes);
send(reply);//sending message response
} catch (Exception e) {JOptionPane.showMessageDialog(null, e.getMessage());}}
```

Query execution and data extraction

Figure 4.20. Code for receiving/responding to request for student exam request

4.6.4 **Providing extra time in an examination**

Teacher gives extra time for an examination via the ‘view student request’ page. Once extra time is given by the teacher, the teacher assistant agent sends this information to the Admin info extract agent on the server who then inserts the information (extra time given, time of issue etc) into the database (Extra_time table, section 3.4).

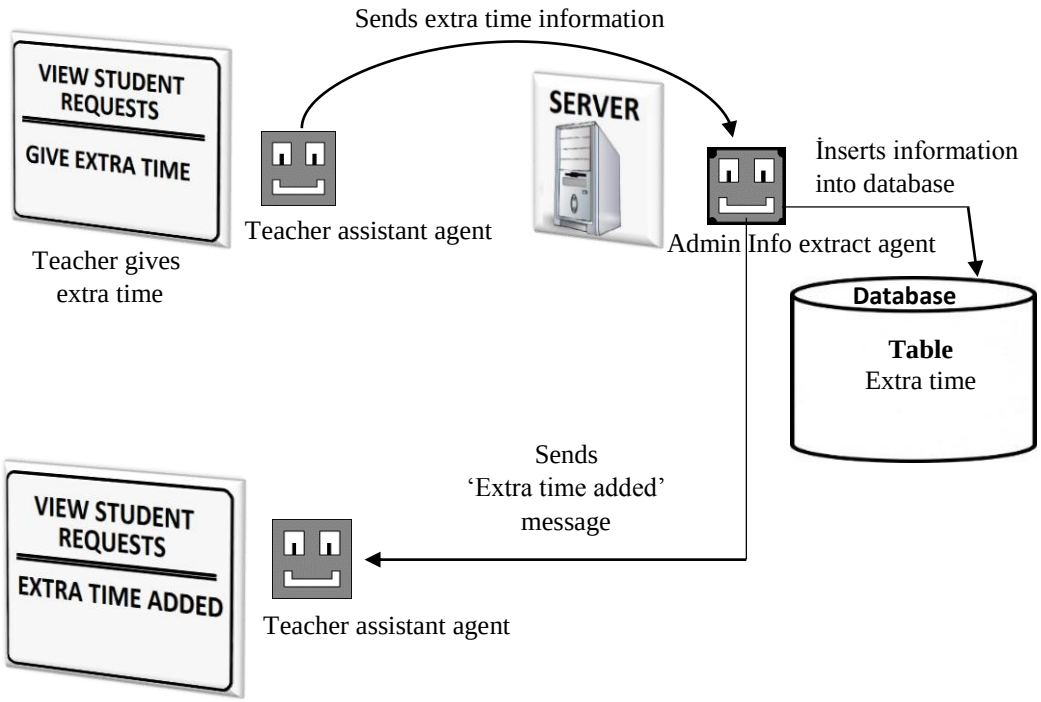


Figure 4.21. System design for setting extra time

```
System.out.println("Agent " + getLocalName() + ": sending REQUEST message...");

ACLMessage request = new ACLMessage(ACLMessage.REQUEST);

AID aid = new AID("AdminInfoExtractAgent@192.168.5.35:1099/JADE", AID.ISGUID); message receiver (AdminInfoExtractAgent)

aid.addAddresses("http://192.168.5.35:7778/acc");// receiver address

request.addReceiver(aid);

request.setOntology("InsertExtratime");

request.setContent(TeacherOptions.Course + "~" + TeacherOptions.eType + "~" + TeacherOptions.rqust.extraTime);// message content(extra time info)

send(request);// sending request message for inserting extra time

System.out.println("Agent sent message " + request.getContent());

myAgent.addBehaviour(recieveinsertExtratimeRequestResponse);
```

Figure 4.22. Code for sending extra time insertion request

```
if (msg.getOntology().equals("InsertExtratime")) {
String message = msg.getContent();// getting message content
messageArray = new String[3];
messageArray = message.split("~");// splitting message into different parts
System.out.println("Agent " + getLocalName() + " recieved REQUEST message " + message);
try {Class.forName("com.mysql.jdbc.Driver");
Connection con = (Connection) //creating database connection
DriverManager.getConnection("jdbc:mysql://localhost:3306/onlineexamination",
"root", "ghsm4/24/7/*");
Statement stmt = con.createStatement();
DateFormat dtform = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
java.util.Date dtobj = new java.util.Date();
String sql = "insert into extra_time values('" + messageArray[0] + "','" + messageArray[1] + "',"
+ Integer.parseInt(messageArray[2]) + ",'" + dtform.format(dtobj) + "')"; // extra time DB
insertion
stmt.executeUpdate(sql);
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());
reply.setOntology("ExtratimeResponse");
reply.setContent("Extra Time was added Sucessfully"); // setting message content
System.out.println("Agent " + getLocalName() + " sending INFORM message " +
reply.getContent());
send(reply);// sending 'reply successful' message
} catch (Exception e) {ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());
reply.setOntology("ExtratimeResponse");
reply.setContent("Extra Time has already been added for this course!");// setting message
content
System.out.println("Agent " + getLocalName() + " sending INFORM message " +
reply.getContent());
send(reply);// sending reply when extra time has alredy been given}}
```

Figure 4.23. Code for receiving/responding to extra time insertion request

Once a student sends an extra time request and receives an 'extra time request was successfully sent' message, the Exam monitor agent periodically checks (every 10 seconds) if the teacher has responded to the student's request by sending a message to the Info extract agent asking if the time has been given. The Info extract agent then queries the database (Extra time table, section 3.4) to determine if extra time has been given.

If extra time has been granted by the teacher, the information is passed across to the Exam monitor agent who then adds this extra time to the remaining minutes of the exam and displays how many minutes has been added to the student.

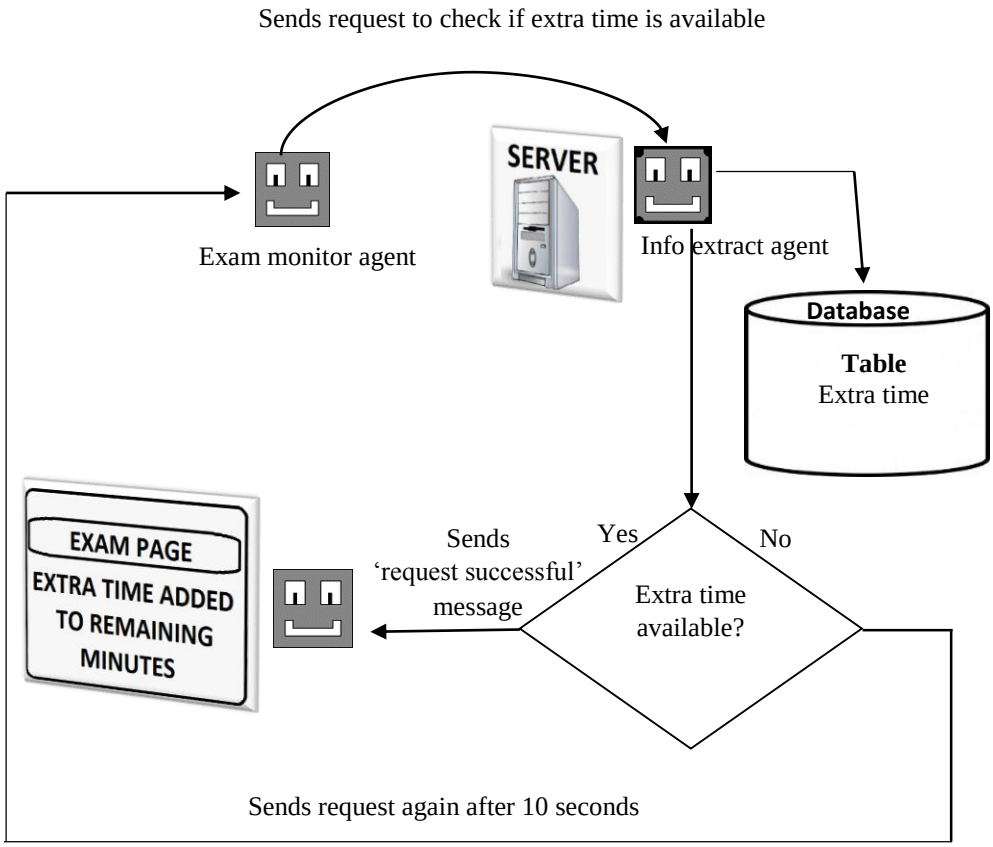


Figure 4.24. System design for adding extra time to exam duration

```
public Behaviour extratimeRequest = new TickerBehaviour(this, 10000) {
protected void onTick() {
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);// request message
AID aid = new AID("InfoExtractAgent@192.168.5.35:1099/JADE", AID.ISGUID);//receiver
aid.addAddresses("http://192.168.5.35:7778/acc");// receiver's address
request.addReceiver(aid);// adding receiver
request.setOntology("GetExtratimeRequest");
request.setContent(StartPage.Course + "~" + Exam_type);// setting message content
System.out.println("Agent " + getLocalName() + ": sending Extra Time REQUEST
message... " + StartPage.Course + "~" + Exam_type);
send(request);// sending request
if (addChecker == 0) {// flag checking if extra time has been received before
myAgent.addBehaviour(receiveExtratime);
addChecker = 1;}}}; // flag for ensuring that extra time is received only once
```

Figure 4.25. Code for checking if extra time is available

```
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);// response message
reply.addReceiver(msg.getSender());// adding sender
reply.setOntology("Extratime");
try {Class.forName("com.mysql.jdbc.Driver");Connection con = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/onlineexamination",
"root", "ghsm4/24/7/*");// creating database connection
Statement stmt = con.createStatement();
String sql = "select Extra_time_added \n"
+ " from  extra_time\n" + " where Course_ID = '" + messageArray[0] + "'\n"
+ " and ExamType = '" + messageArray[1] + "'";// query for getting extra time
ResultSet rs = stmt.executeQuery(sql);// executing query
int extratime = 0;
while (rs.next()) {extratime = rs.getInt("Extra_time_added");}
if (extratime > 0) {reply.setContent(""+extratime); // setting message content
System.out.println("Agent " + getLocalName() + " sending INFORM message Extra Time :"
+ extratime);
send(reply);}} catch (Exception e){JOptionPane.showMessageDialog(null,e.getMessage());}}
```

Figure 4.26. Code for responding to extra time request

```
if (extra_timeChecker == 1) {// flag for checking if extra time has been added
deactivation_Minutes = deactivation_Minutes + extra_time;
ExamPage.extratimeAddedLabel.setForeground(Color.red);
ExamPage.extratimeAddedLabel.setText("Extra Time (" + extra_time + "
mins) has been added");
ExamPage.extratimeAddedLabel.setVisible(true);
extra_timeChecker = 2;}//flag to prevent extra time from being added again
```

Figure 4.27. Code for adding extra time to exam duration

The various illustrations depicted in this section helps to give an idea of how the system was designed and implemented, how the agents work/communicate together and how information from the database is queried and extracted by agents.

Chapter 5

DISCUSSIONS

In this section the issues and challenges encountered while implementing this thesis and the measures taken to solve them is discussed. Some of the challenges faced are listed as follows:

a) **Defining, Structuring and Programming the Modules of the System:** For the system to perform its functions in the most efficient way the modules of the system have to be well defined, well-structured and properly programmed. To solve the issues behind this, the problem that the system is aimed at solving was first of all defined. Secondly, the various steps and operations that can be taken to solve this problem was also defined and the best alternative was selected. The steps are then divided into different interconnected modules and the structure of the system is finally constructed. Once the structure of the system is put in place a great deal was put into programming the system. Various test and trials were done at each level of programming and unnecessary parts were eliminated to produce the final results.

b) **Creating a Structured Database:** When handling important information such as that related to student exams and results it is very important to carefully design a database where the necessary information can be adequately stored and easily retrieved when queried. The challenges in designing a well-structured database was resolved by analysing the system functions, the users of the system and the

type of data that requires storage. With this information the adequate tables were created and connected accordingly.

c) **Defining the Behaviour of the Agents:** The behaviour of an agent plays a very important role in the functionality of the system. It is important to properly define how agents should react in different situations and what they should do with the information they receive. The issues behind designing agent behaviour was solved by properly defining the goal of each agent. Once the goals were defined, the behaviour of the agents was then carefully programmed and designed to meet these goals in the most effective and efficient way possible.

d) **Agent Communication:** In the system, agents assist one another in performing various duties by communicating with one another through messages. The structure of agent messages, and the medium of date transfer between agents are very important factors in agent communication. It is also important to ensure that the messages sent are properly received by the recipient agent in its proper order. To solve this problem, agents were designed to send series of messages as a single string of data. This data is then collected by the receiving agent and divided into its different parts. These parts are then used by the agent to perform its duties. For example, an examination monitor agent can send the performance statistics of a student as a single line of data. The receiving agent on the student's PC takes the data, breaks it down into different parts and uses the resulting information to design a performance chart which is displayed to the student.

e) **Defining the Proper Variables:** In order for agents and the system as a whole to function efficiently, the proper parameters have to be initialized. It is important to define which variable should be public, private, static etc.; what types these variables should be (double, char, float etc.), and how they should be used. Determining the right variables to use was achieved by analysing the type of operations performed on the system and the type of data handled by the system.

f) **Creating a User-riendly System**: A user-friendly system can be quite tasking to create since the preferences of various users differ. The best solution found to tackle this problem was to simplify the outlook of the system as much as possible and provide features that are useful to the users of the system.

Chapter 6

CONCLUSION AND FUTURE WORKS

Examinations are an important part of our society and creating an evaluation system with smart, helpful, user-friendly features will greatly impact people's learning capabilities. This thesis aims at providing these features with use of intelligent multi-agent systems.

The paper-based examination (PBE) system is the most common medium of evaluation and useful in many ways but limitations that inhibit its efficiency have been detected. It is prone to human errors, human limitations, and delays, it is bulky tedious and time consuming, it is costly with high security risks, and there is rigidity in exam time and location. With the growing rate of internet and computer technologies an online agent-based alternative was proposed in this thesis.

Using Netbeans and JADE multi-agent framework the exam system was implemented. The complex tasks of the system was broken down into modules/subtasks handled by specific agents designed to achieve certain goals. The system eliminates the delays and limitations of the PBE system by providing automated services such as instantaneous display of results, monitory and handling of exams by agents (agents activate and deactivate exams , calculate estimated time to finish, displays exam questions and exam scripts , provides extra time if given by teachers etc.), providing detailed analysis

of a student performance (weak points and strong points) and displaying improvement percentage. The use of a mult-agent framework opens doors for intelligence or smartness to be introduced or added into the system; simplifies and improves the way online exam systems are handled and enhances the efficiency of online.

The major limitation of the system is the security of the agents and the data passed across between agents i.e. providing mediums for preventing security attacks on agents or illegal interceptions of data during agent communication. This limitation will be dealt with as further work is done on the system. Other future works to be done on this thesis is to broaden the scope of the thesis from standard university exams to distance learning exams; improve the security and safety of the system; increase interactions between users and agents; improve the outlook of the system, and provide more helpful features such as bulk loading of data (e.g. exam questions when setting exams) and exporting of data (e.g. exam results and attendance) into files for storage or printing.

REFERENCES

[1] C. P. Newhouse, "Computer-based Exams in Schools: Freedom from the Limitations of Paper?," *Research and Practice in Technology Enhanced Learning,* vol. 8, no. 3, pp. 431-444, 2013.

[2] R. D. Gawali and B. B. Meshram, "Agent-based Autonomous Examination Systems," in *Intelligent Agent & Multi-Agent Systems*, 2009.

[3] B. Hang, "The Design and Implementation of Online Examination System," in *Computer Science and Society (ISCCS), 2011 International Symposium*, 2011.

[4] D. Xiaoyu and L. Yunhao, "Research and Implementation of Web-Online English Testing System," in *Information Science and Engineering (ISISE), 2010 International Symposium*, 2010.

[5] A. Ahmad, N. U. Khan and A. W. Abbas, "PHP+MySQL based Online Examination System with Power Failure Handling and Dropbox Capability," in *Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference*, 2013.

[6] P. Guo, H.-f. Yu and Q. Yao, “The Research and Application of Online Examination and Monitoring system,” in *IT in Medicine and Education, ITME 2008.*, 2008.

[7] S. Wei-feng, H. Meng and L. Jun, “An Online Examination System Supporting User-defined Question Type,” in *Education Technology and Computer (ICETC), 2010 2nd International Conference*, 2010.

[8] X. Li and Y. Wu, “Design and Development of the Online Examination and Evaluation System Based on B/S Structure,” in *Wireless Communications, Networking and Mobile Computing, WiCom 2007*, 2007.

[9] C. Liu and K. Wang, “An Online Examination System based on UML Modeling and MVC Design Pattern,” in *Control Engineering and Communication Technology (ICCECT), 2012 International Conference*, 2012.

[10] H. Lu and Y. Hu, “The Design and Implementation of Online Examination System based on J2EE,” in *Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference*, 2012.

[11] L. Jun, “Design of Online Examination System Based on Web Service and COM,” in *Information Science and Engineering (ICISE), 2009 1st International Conference*, 2009.

[12] S. Murugesan, “Intelligent Agents on the Internet and Web,” in *1998 IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control*, 1998.

[13] T. Magedanz, K. Rothermel and S. Krause, “Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?,” in *Networking the Next Generation. Fifteenth Annual Joint Conference of the IEEE Computer Societies.*, 1996.

[14] F. L. Bellifemine, G. Caire and D. Greenwood, Developing Multi-Agent Systems with JADE, John Wiley & Sons, 2007.

[15] N. S. Bhuvaneswari, S. Sujatha and C. Deepthi, “Proposal on Centralized Exam Assessment Apportion Using Mobile Agents,” in *Communication and Computational Intelligence (INCOCCI), 2010 International Conference*, 2010.

[16] Y. H. Ali and Z. F. Hussain, “Smart Electronic Examination System using Multi-Agent Platform,” *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS),* vol. 3, no. 1, pp. 42-46, 2014.

[17] M. Wooldridge, “Intelligent Agents,” in *Multiagent Systems,* G. Weiss, Ed., The MIT Press, 2000, pp. 27-73.

[18] “Wikipedia,” Wikimedia Foundation, 20 December 2014. [Online]. Available: http://en.wikipedia.org/wiki/Intelligent_agent. [Accessed 7 February 2015].

[19] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2003.

[20] G. Caire, F. Bellifemine, A. Poggi and G. Rimassa, “JADE: A White Paper,” 2003.

[21] B. Hermans, “Intelligent Software Agents on the Internet,” 11 July 2000. [Online]. Available: http://www.hermans.org/agents/index.html. [Accessed 8 February 2015].

[22] D. Gilbert, *Intelligent Agents: The Right Information at the Right Time,* IBM Intelligent Agent White Paper, 1997.

[23] H. Tran and T. Tran , “Intelligent Agent,” [Online]. Available: http://groups.engin.umd.umich.edu/CIS/course.des/cis479/projects/agent/Intelligent_agent.html.

[24] “Wikipedia,” Wikimedia Foundation, 21 January 2015. [Online]. Available: http://en.wikipedia.org/wiki/Multi-agent_system.

[25] “Multi-Agent Systems,” The Robotics Institute - Carnegie Mellon University, 2006 - 2012. [Online]. Available: http://www.cs.cmu.edu/~softagents/multi.html. [Accessed 2015 February 8].

[26] K. J. Mackin and E. Tazaki, “Evolving Intelligent Multiagent Systems using Unsupervised Agent Communication,” in *Systems, Man, and Cybernetics, 2000 IEEE International Conference*, 2000.

[27] Z. Guessoum, “Adaptive Agents and Multiagent Systems,” 2004.

[28] D. Weyns and M. Georgeff, “Self-Adaptation Using Multiagent Systems,” 2010.

[29] “JaCaMo Project,” WordPress, [Online]. Available: http://jacamo.sourceforge.net/?page_id=40. [Accessed February 2014].

[30] K. P. Sycara, “Multiagent,” *AI Magazine,* vol. 19, pp. 79-92, 1998.

[31] C.-i. Peña, J.-l. Marzo and J.-l. Rosa, “Intelligent Agents in a Teaching and Learning Environment on the Web,” in *International Conference on Advanced Learning Technologies*, 2002.

[32] “Cougaar Software Inc.,” Cougaar Software Inc., [Online]. Available: http://www.cougaarsoftware.com/agents/agents-1.htm.

[33] I. Rudowsky, “Intelligent Agents,” in *The Americas Conference on Information Systems*, 2004.

[34] P. Stone, “Why Multiagent Systems?,” 1996.

[35] G. Caire, “JADE Programming for Beginners,” 2009.

[36] F. Bellifemine, A. Poggi and G. Rimassa, “JADE: A FIPA Compliant Agent Framework”.

[37] D. Grimshaw, “JADE Administration Tutorial,” 26 March 2010. [Online]. Available: http://jade.tilab.com/documentation/tutorials-guides/jade-administration-tutorial/.

APPENDICES

Appendix A: Java program for Login agent on the server-side.

```
package my.onlineexam;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.nio.charset.Charset;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import javax.swing.JOptionPane;
public class LoginAgent extends Agent {
String[] messageArray;
//Message template for sending message to login agent
private final MessageTemplate template = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.REQUEST),
MessageTemplate.MatchOntology("Login"));
protected void setup() { // Agent startup activities
System.out.println("Agent " + getLocalName() + " is ready.");
 // Cyclic behaviour of login user agent
addBehaviour(new CyclicBehaviour(this) {
public void action() {// actions of the login user agent is defined below
ACLMessage msg = myAgent.receive(template); // receives message
if (msg != null) {
String message = msg.getContent();// saves message content in message string
System.out.println("Agent " + getLocalName() + " recieved REQUEST message");
System.out.println("The message is " + message);
messageArray = new String[3];
messageArray = message.split("~");// splits message and saves it in a message array
try {Class.forName("com.mysql.jdbc.Driver");// creatating db connection
Connection con = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/onlineexamination",
"root", "ghsm4/24/7/*");
Statement stmt = con.createStatement();
if (messageArray[2].equals("Student")) {
// Query for getting student information from database
String sql = "Select student_id,student_name,"
+ "student_surname,student_password "+ "from student where student_id='" +
messageArray[0] + "'and student_password='" + messageArray[1] + "'";
ResultSet rs = stmt.executeQuery(sql);
```

```
if (rs.first()) {// Extracting the Query results
String sid = rs.getString("student_id");
String spswd = rs.getString("student_password");
String sn = rs.getString("student_name");
String ssn = rs.getString("student_surname");
if (sid.equals(messageArray[0]) && spswd.equals(messageArray[1])) {
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());
System.out.println("Agent " + getLocalName() + " sending INFORM message."
+ "Valid~" + sid + "~" + sn + "~" + ssn);
reply.setOntology("Login");
//Setting 'valid user' message content
reply.setContent("Valid~" + sid + "~" + sn + "~" + ssn);
send(reply);//sending valid message }
} else {ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());// setting message receiver
reply.setOntology("Login");
//Setting 'invalid user' message content
reply.setContent("Invalid~Invalid~Invalid~Invalid");
System.out.println("Agent " + getLocalName() + " sending INFORM message");
send(reply);//sending invalid message}}
if (messageArray[2].equals("Admin")) {
// Query for getting teacher information from database
Stringsql="SelectAdmin_username,Admin_name,Admin_password,Admin_surnam"
+ "from administration where Admin_username='" + messageArray[0]
+ "'and Admin_password='" + messageArray[1] + "'";
ResultSet rs = stmt.executeQuery(sql);
if (rs.first()) {// Extracting the Query results
String ausid = rs.getString("Admin_username");
String spswd = rs.getString("Admin_password");
String aname = rs.getString("Admin_name");
String asname = rs.getString("Admin_surname");
if (ausid.equals(messageArray[0]) && spswd.equals(messageArray[1])) {
ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());
reply.setOntology("Login");
//Setting 'valid user' message content
reply.setContent("Valid~" + ausid + "~" + aname + "~" + asname);
System.out.println("Agent " + getLocalName() + " sending INFORM message"
+ "Valid~" + ausid + "~" + aname + "~" + asname);
send(reply);//sending valid message}
} else {ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
reply.addReceiver(msg.getSender());
reply.setOntology("Login");
```

```
//Setting 'invalid user' message content
reply.setContent("Invalid~Invalid~Invalid~Invalid");
System.out.println("Agent " + getLocalName() + " sending INFORM message");
send(reply);//sending invalid message}}
} catch (Exception e) {JOptionPane.showMessageDialog(null, e.getMessage());}
} else {block();}}});}
protected void takeDown() {
System.out.println("Agent terminating....");}}
```

Appendix B: Java program for Login agent on the server-side.

```
package my.onlineexam;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.awt.Color;
public class LoginUserAgent extends Agent {
public static LoginPort LoginGui;
public static SplashScreen splash; public static String UserId;
public static String name_of_user; public static String surname_of_user;
public static int loadValue = 0;
//TeacherOptions Topts = new TeacherOptions();
public static String message; String[] messageArray;
//Message template for sending message to login user agent
private final MessageTemplate template = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("Login"));
//user-defined agent behaviour
public Behaviour send = new Behaviour() {
public void action() {// actions of the login user agent is defined below
System.out.println("Agent " + getLocalName() + ": sending REQUEST message...");
//creating a request message
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
//creating message receiver
AID aid = new AID("LoginAgent@192.168.5.35:1099/JADE", AID.ISGUID);
//setting receiver address
aid.addAddresses("http://192.168.5.35:7778/acc");
//adding receiver
request.addReceiver(aid);
//setting the message ontology
request.setOntology("Login");
//Setting message content information entered by user
request.setContent(LoginGui.username+"~"+LoginGui.Pswd+"~"+LoginGui.userTy
pe);
//sending message
send(request);
System.out.println("Agent " + getLocalName() + ": sent REQUEST message..." +
LoginGui.username + "-" + LoginGui.Pswd + "-" + LoginGui.userType);
//adding behaviour for recieving message response
```

```
myAgent.addBehaviour(receive);}
public boolean done() {return true;// returns when actions are completed}};
//behaviour for receiving message response
public Behaviour receive = new OneShotBehaviour() {
public void action() {
ACLMessage msg = myAgent.blockingReceive(template);//receiving messge
if (msg != null) {message = msg.getContent();// saving message content in message
string
System.out.println(" Agent " + msg.getSender().getName() + " message is " +
message);
messageArray = new String[4];
messageArray = message.split("~");// splits message and saves it in a message array
//adding behaviour for giving access to users
myAgent.addBehaviour(openAcess);
} else {block();}} };
//behaviour for giving access to users
Behaviour openAcess = new OneShotBehaviour() {
public void action() {
// granting access to students when sucessfully validated
if (messageArray[0].equals("Valid") && LoginGui.userType.equals("Student")) {
UserId = messageArray[1];
name_of_user = messageArray[2];
surname_of_user = messageArray[3];
LoginGui.setVisible(false);
StudentAssistantAgent.Stp.jLabel1.setText("Welcome " + name_of_user + " " +
surname_of_user + "!");
StudentAssistantAgent.Stp.CourseCombo.removeAllItems();
StudentAssistantAgent.Stp.FillCombo();
StudentAssistantAgent.Stp.setVisible(true);}
//denying access to invalid users
if (messageArray[0].equals("Invalid")) {
LoginGui.loginMessageText.setText("Invalid Username and/or Password and/or
Usertype");
LoginGui.loginMessageText.setVisible(true);}
// granting access to teachers when sucessfully validated
if (messageArray[0].equals("Valid") && LoginGui.userType.equals("Admin")) {
UserId = messageArray[1];
name_of_user = messageArray[2];
surname_of_user = messageArray[3];
LoginGui.setVisible(false);
TeacherAssistantAgent.tops.jLabel3.setText("Welcome " + name_of_user
+ " " + surname_of_user + "!");
TeacherAssistantAgent.tops.courseCombo.removeAllItems();
TeacherAssistantAgent.tops.FillCourseCombo();
```

```
TeacherAssistantAgent.tops.setVisible(true);}}};
// behaviour for displaying splashscreen at the start of the system
public Behaviour splashScreenTimer = new TickerBehaviour(this, 200) {
protected void onTick() {
if (loadValue == 101) {
splash.close();
LoginGui.setVisible(true);
myAgent.removeBehaviour(splashScreenTimer);}
if (loadValue < 101) {SplashScreen.splashScreenProgressBar.setValue(loadValue);
loadValue = loadValue + 1;}}};
protected void setup() {// Agent startup activities
splash = new SplashScreen();
this.addBehaviour(splashScreenTimer);
splash.setVisible(true);
LoginGui = new LoginPort(this);
System.out.println("Online Exam System Started....");
System.out.println("LoginUserAgent is ready");
}protected void takeDown() {System.out.println("Agent terminating....");}}
```

Appendix C: Java program for the Exam monitor agent

```
package my.onlineexam;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.awt.Color;
import java.text.DateFormat;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
public class ExamMonitorAgent extends Agent {
//variable declarations
public static ExamPage Exp;public static ResultPerformancePage resultPage;
public static String message; public static int Duration; public static double
estimated_Time; public static int halfTime; public static int littleTime;
public static int Question_number = 0; public static int extra_time;
public static int extra_timeChecker = 0; public static int addChecker = 0;
public static String Exam_number; public static String Exam_type;public static
String Question_ID;public static String Category;public static String[]
messageArray;public static String[] optionsArray;public static String[]
StatsArray;public static int grade;public static double initial_Minutes;public static
double current_Minutes;public static int deactivation_Minutes;
//Message templates for sending messages
private final MessageTemplate initExamTemplate = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("InitExamQuestions"));
private final MessageTemplate examTemplate = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("ExamQuestions"));
private final MessageTemplate optionsTemplate = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("Options"));
```

```
private final MessageTemplate unansTemplate = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("UnansInfo"));
private final MessageTemplate gradeTemplate = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("GradeInfo"));
private final MessageTemplate statTemplate = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("StatInfo"));
private final MessageTemplate extraTime = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("Extratime"));
private final MessageTemplate noOfQuestionsTemplate = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("QuestionsStatistics"));
private final MessageTemplate StudentRequestResponse = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("StudentRequestResponse"));
private final MessageTemplate examScriptTemplate = MessageTemplate.and(
MessageTemplate.MatchPerformative(ACLMessage.INFORM),
MessageTemplate.MatchOntology("ExamScript"));
//behavior for sending initial exam information request
public Behaviour init_sendExamRequest = new OneShotBehaviour() {
public void action() {
//System.out.println("Agent " + getLocalName() + ": sending REQUEST
message...");
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("GetInitExamQuestions");
Question_number++;// setting question number
ExamPage.questionNoLabel.setVisible(true);
ExamPage.questionNoLabel.setText(Question_number + ".");
request.setContent(LoginUserAgent.UserId + "~" + StartPage.Course + "~" +
Question_number);
send(request);// sending request
System.out.println("Agent " + getLocalName()
+ ": sent REQUEST message..." + LoginUserAgent.UserId + "~"
+ StartPage.Course + "~" + Question_number);
myAgent.addBehaviour(init_recieveExamInfo);}};
//behavior for receiving initial exam information
public Behaviour init_recieveExamInfo = new CyclicBehaviour() {
public void action() {// action method for agent behaviour
```

```
int i;
ACLMessage msg = myAgent.receive(initExamTemplate);
if (msg != null) {
message = msg.getContent();
System.out.println(" Agent " + msg.getSender().getName() + " INIT RECIEVE
message is " + message);
messageArray = new String[7];
messageArray = message.split("~");
if (messageArray[0].equals("Valid")) {
//current time gotten to calculate initial remaining time
DateFormat dtform1 = new SimpleDateFormat("HH:mm:ss");
Date dtobj1 = new Date();
String dateStr1 = dtform1.format(dtobj1);
System.out.println("Length: " + dateStr1.length() + " Initial Hour: " +
dateStr1.substring(0, 2) + " Minutes: " + dateStr1.substring(3, 5) + " Seconds: " +
dateStr1.substring(6, 8));
int chour = Integer.parseInt(dateStr1.substring(0, 2));
int cmin = Integer.parseInt(dateStr1.substring(3, 5));
int cseconds = Integer.parseInt(dateStr1.substring(6, 8));
deactivation_Minutes = Integer.parseInt(messageArray[3]);
current_Minutes = (chour * 60) + (cmin) + ((1 / (double) 60) * cseconds);
double initial_Time = deactivation_Minutes - current_Minutes;
DecimalFormat dff = new DecimalFormat("#");
String initial_TimeStr = dff.format(initial_Time);
System.out.println("The initial current time in mins: " + current_Minutes +
"Deactivation mins: " + deactivation_Minutes);
ExamPage.checker = 0;
ExamPage.QuestionLabel.setText(messageArray[4]);
ExamPage.TimeLabel.setText(initial_TimeStr);
ExamPage.QuestionLabel.setVisible(true);
ExamPage.TimeLabel.setVisible(true);
ExamPage.minsLabel.setVisible(true);
ExamPage.NextButton.setVisible(true);
ExamPage.NextButton.setEnabled(true);
ExamPage.SubmitButton.setVisible(true);
ExamPage.requestTeacher.setVisible(true);
ExamPage.UnansweredButton.setVisible(true);
ExamPage.unAnsNext.setVisible(true);
ExamPage.requestExtratime.setVisible(true);
ExamPage.qUnansLAbel.setVisible(true);
ExamPage.noOfUnansLabel.setVisible(true);
ExamPage.qAnsLabel.setVisible(true);
ExamPage.QuestAnsLabel.setVisible(true);
ExamPage.estimateLabel.setVisible(true);
```

```
ExamPage.estimatedTimeLabel.setVisible(true);
Exam_number = messageArray[1];
Exam_type = messageArray[6];
Duration = Integer.parseInt(initial_TimeStr);
// initial_Duration = Duration;
ExamPage.estimatedTimeLabel.setText("" + Duration);
DecimalFormat df = new DecimalFormat("#");
String halfTimeStr = df.format(Duration * 0.7);
halfTime = Integer.parseInt(halfTimeStr);
littleTime = Duration - (int) (Duration * 0.8);
System.out.println("Halftime: " + halfTime);
Question_ID = messageArray[2];
Category = messageArray[5];
myAgent.addBehaviour(sendOptionsRequest);
myAgent.addBehaviour(sendQuestionsStatistics);
myAgent.addBehaviour(Timekeeper);
DateFormat dtform = new SimpleDateFormat("HH:mm:ss");
Date dtobj = new Date();
String dateStr = dtform.format(dtobj);
System.out.println("Length: " + dateStr.length() + " Initial Hour: " +
dateStr.substring(0, 2) + " Minutes: " + dateStr.substring(3, 5) + " Seconds: " +
dateStr.substring(6, 8));
int hour = Integer.parseInt(dateStr.substring(0, 2));
int min = Integer.parseInt(dateStr.substring(3, 5));
int seconds = Integer.parseInt(dateStr.substring(6, 8));
/*Initial min for estimated time( Has to be after options have been sent and recived
and time has been set that's why with used two different initial mins instead of using
just one)*/
initial_Minutes = (hour * 60) + (min) + ((1 / (double) 60) * seconds);
System.out.println("The time in mins: " + initial_Minutes);}
if (messageArray[0].equals("Invalid")) {
ExamPage.QuestionLabel.setVisible(true);
ExamPage.QuestionLabel.setText(" YOU HAVE ALREADY TAKEN THIS EXAM
OR EXAMINATION IS NOT AVAILABLE");
ExamPage.UnansweredButton.setVisible(false);
ExamPage.unAnsNext.setVisible(false);
ExamPage.NextButton.setVisible(false);
ExamPage.OptionA.setVisible(false);
ExamPage.OptionB.setVisible(false);
ExamPage.OptionC.setVisible(false);
ExamPage.OptionD.setVisible(false);
ExamPage.OptionE.setVisible(false);
ExamPage.SubmitButton.setVisible(false);
ExamPage.requestExtratime.setVisible(false);
```

```
ExamPage.requestTeacher.setVisible(false);
ExamPage.questionNoLabel.setVisible(false);
ExamPage.outOfTimeImage.setVisible(false);
ExamPage.outOfTimeLabel.setVisible(false);
ExamPage.qUnansLAbel.setVisible(false);
ExamPage.noOfUnansLabel.setVisible(false);
ExamPage.qAnsLabel.setVisible(false);
ExamPage.QuestAnsLabel.setVisible(false);
ExamPage.estimateLabel.setVisible(false);
ExamPage.estimatedTimeLabel.setVisible(false);
ExamPage.TimeLabel.setVisible(false);
ExamPage.minsLabel.setVisible(false);
ExamPage.checker = 1;}
} else {block();}}};
//behaviour for sending exam information request
public Behaviour sendExamRequest = new OneShotBehaviour() {
public void action() {
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("GetExamQuestions");
Question_number++;
ExamPage.questionNoLabel.setVisible(true);
ExamPage.questionNoLabel.setText(Question_number + ".");
request.setContent(LoginUserAgent.UserId + "~" + StartPage.Course + "~" +
Question_number);
send(request);
System.out.println("Agent " + getLocalName() + ": sent REQUEST message..." +
LoginUserAgent.UserId + "~" + StartPage.Course + "~" + Question_number);
myAgent.addBehaviour(recieveExamInfo);        }};
// behaviour for receiving exam information
public Behaviour recieveExamInfo = new CyclicBehaviour() {
public void action() {int i;
ACLMessage msg = myAgent.receive(examTemplate);
if (msg != null) {message = msg.getContent();
System.out.println(" Agent " + msg.getSender().getName() + " message is " +
message);
messageArray = new String[5];
messageArray = message.split("~");
if (messageArray[0].equals("Valid")) {
ExamPage.QuestionLabel.setText(messageArray[3]);
Question_ID = messageArray[2];
Category = messageArray[4];
```

```
ExamPage.NextButton.setVisible(true);
ExamPage.SubmitButton.setVisible(true);
ExamPage.NextButton.setEnabled(true);
myAgent.addBehaviour(sendOptionsRequest);}
if (messageArray[0].equals("Invalid")) {
ExamPage.QuestionLabel.setVisible(true);
ExamPage.QuestionLabel.setText("There are no Further Questions "
+ "pls click the  \"Unanswered questions\" button to complete unanswered questions
"+ "or click Submit to Finish Exam");
ExamPage.UnansweredButton.setEnabled(true);
ExamPage.NextButton.setEnabled(false);
ExamPage.OptionA.setVisible(false);
ExamPage.OptionB.setVisible(false);
ExamPage.OptionC.setVisible(false);
ExamPage.OptionD.setVisible(false);
ExamPage.OptionE.setVisible(false);
ExamPage.questionNoLabel.setVisible(false);
ExamPage.outOfTimeImage.setVisible(false);
ExamPage.outOfTimeLabel.setVisible(false);}
} else {block();}}};
// behaviour for sending request for exam question options
public Behaviour sendOptionsRequest = new OneShotBehaviour() {
public void action() {System.out.println("Agent " + getLocalName() + ": sending
OPTIONS REQUEST message...");
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("GetOptions");
request.setContent(Question_ID);
send(request);
System.out.println("Agent " + getLocalName() + ": sent OPTIONS REQUEST
message...");
myAgent.addBehaviour(recieveOptionsInfo);}};
// behaviour for receiving exam question options
public Behaviour recieveOptionsInfo = new CyclicBehaviour() {
public void action() {
int i;
ACLMessage msg = myAgent.receive(optionsTemplate);
if (msg != null) {
message = msg.getContent();
System.out.println(" Agent " + msg.getSender().getName() + " message is " +
message);
optionsArray = new String[5];
```

```
optionsArray = message.split("~");
ExamPage.OptionA.setText(optionsArray[0]);
ExamPage.OptionA.setVisible(true);
ExamPage.OptionB.setText(optionsArray[1]);
ExamPage.OptionB.setVisible(true);
ExamPage.OptionC.setText(optionsArray[2]);
ExamPage.OptionC.setVisible(true);
ExamPage.OptionD.setText(optionsArray[3]);
ExamPage.OptionD.setVisible(true);
ExamPage.OptionE.setText(optionsArray[4]);
ExamPage.OptionE.setVisible(true);
} else {block();}} };
// behaviour for sending request for unanswered questions
public Behaviour sendUnansRequest = new OneShotBehaviour() {
public void action() {
System.out.println("Agent " + getLocalName() + ": sending UNANS REQUEST
message...");
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("GetUnansExamQuestions");
request.setContent(LoginUserAgent.UserId + "~" + Exam_number);
send(request);
System.out.println("Agent " + getLocalName() + ": sent REQUEST message..." +
LoginUserAgent.UserId + "~" + Exam_number);
myAgent.addBehaviour(recieveUnansInfo);}};
// behaviour for receiving unanswered question information
public Behaviour recieveUnansInfo = new CyclicBehaviour() {
public void action() {
int i;
ACLMessage msg = myAgent.receive(unansTemplate);
if (msg != null) {
message = msg.getContent();
System.out.println(" Agent " + msg.getSender().getName() + " message is " +
message);
messageArray = new String[5];
messageArray = message.split("~");
if (messageArray[0].equals("Valid")) {
ExamPage.unAnsNext.setEnabled(true);
ExamPage.QuestionLabel.setText(messageArray[1]);
ExamPage.QuestionLabel.setVisible(true);
Question_ID = messageArray[2];
Category = messageArray[4];
```

```
ExamPage.questionNoLabel.setVisible(true);
ExamPage.questionNoLabel.setText(messageArray[3]);
myAgent.addBehaviour(sendOptionsRequest);}
if (messageArray[0].equals("Invalid")) {
ExamPage.QuestionLabel.setVisible(true);
ExamPage.QuestionLabel.setText("There are no Further Questions pls Click Submit
to Finish Exam");
ExamPage.UnansweredButton.setEnabled(false);
ExamPage.unAnsNext.setEnabled(false);
ExamPage.OptionA.setVisible(false);
ExamPage.OptionB.setVisible(false);
ExamPage.OptionC.setVisible(false);
ExamPage.OptionD.setVisible(false);
ExamPage.OptionE.setVisible(false);
ExamPage.requestExtratime.setEnabled(false);
ExamPage.requestTeacher.setEnabled(false);
ExamPage.questionNoLabel.setVisible(false);
ExamPage.outOfTimeImage.setVisible(false);
ExamPage.outOfTimeLabel.setVisible(false);
ExamPage.extratimeAddedLabel.setVisible(false);}
} else {block();}}};
// behaviour for sending student selected answerz
public Behaviour sendAnalyseInfo = new OneShotBehaviour() {
public void action() {
System.out.println("Agent " + getLocalName() + ": sending INFORM message...");
ACLMessage inform = new ACLMessage(ACLMessage.INFORM);
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
inform.addReceiver(aid);
inform.setOntology("AnswerInformation");
inform.setContent(Exam_number + "~" + LoginUserAgent.UserId + "~" +
Question_ID + "~" + ExamPage.Letter + "~" + Category);
send(inform);}};
// behaviour for sending exam grade request
public Behaviour sendGradeRequest = new OneShotBehaviour() {
public void action() {
System.out.println("Agent " + getLocalName() + ": sending GRADE REQUEST
message...");
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("GetExamGrade");
request.setContent(Exam_number + "~" + LoginUserAgent.UserId);
```

```
send(request);
System.out.println("Agent " + getLocalName() + ": sent REQUEST message..." +
StartPage.Course + "~" + LoginUserAgent.UserId);
myAgent.addBehaviour(recieveGradeInfo);}};
// behaviour for receiving exam grade
public Behaviour recieveGradeInfo = new CyclicBehaviour() {
public void action() {
ACLMessage msg = myAgent.receive(gradeTemplate);
if (msg != null) {
message = msg.getContent();
System.out.println(" Agent " + msg.getSender().getName() + " message is " +
message);
messageArray = new String[1];
messageArray = message.split("~");
grade = Integer.parseInt(messageArray[0]);
if (ExamPage.checker == 2) {
ExamPage.Rp.ExamFinishedLabel.setText("Examination Time has Elapsed " +
StartPage.Course + " Exam is Over");}
if (ExamPage.checker == 0) {
ExamPage.Rp.ExamFinishedLabel.setText("You have sucessfully finished " +
StartPage.Course);}
ExamPage.Rp.PStatButton.setVisible(true);
ExamPage.Rp.ExamGradeLabel.setText("Your score is " + grade);
ExamPage.Rp.ExamFinishedLabel.setVisible(true);
ExamPage.Rp.ExamGradeLabel.setVisible(true);
myAgent.removeBehaviour(Timekeeper);
} else {block();
}} };
// behaviour for sending exam performance statistics request
public Behaviour sendStatisticsRequest = new OneShotBehaviour() {
public void action() {
System.out.println("Agent " + getLocalName() + ": sending STATISTICS
REQUEST message...");
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("GetExamStats");
request.setContent(LoginUserAgent.UserId + "~" + Exam_number);
send(request);
System.out.println("Agent " + getLocalName() + ": sent REQUEST message..." +
LoginUserAgent.UserId + "~" + Exam_number);
myAgent.addBehaviour(recieveStatInfo);} };
 // behaviour for receiving exam performance statistics information
```

```
public Behaviour recieveStatInfo = new CyclicBehaviour() {
public void action() {
ACLMessage msg = myAgent.receive(statTemplate);
if (msg != null) {
message = msg.getContent();
System.out.println(" Agent " + msg.getSender().getName() + " message is " +
message);
String length = message.substring(0, message.indexOf('~'));
message = message.substring(message.indexOf('~') + 1);
StatsArray = new String[Integer.parseInt(length)];
StatsArray = message.split("~");
DefaultCategoryDataset dataset = new DefaultCategoryDataset();
int i;
for (i = 0; i < StatsArray.length; i = i + 2) {
dataset.setValue(Integer.parseInt(StatsArray[i + 1]), "grade", StatsArray[i]);}
JFreeChart chart = ChartFactory.createBarChart3D("Performance Statistics",
"Category", "Grade(%)", dataset, PlotOrientation.VERTICAL, false, true, false);
//chart.setBackgroundPaint(Color.BLUE);
chart.getTitle().setPaint(Color.BLUE);
CategoryPlot P = chart.getCategoryPlot();
P. (Color.BLACK);
ChartFrame cFrame = new ChartFrame("Performance Statistics Chart", chart);
cFrame.setVisible(true);
cFrame.setSize(700, 500);
cFrame.setLocationRelativeTo(null);
} else {block();}  }};
// behaviour for sending student request
public Behaviour sendStudentRequest = new OneShotBehaviour() {
public void action() {
System.out.println("Agent " + getLocalName() + ": sending REQUEST message...");
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("InfoExtractAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("InsertStudentRequest");
request.setContent(LoginUserAgent.UserId + "~" + LoginUserAgent.name_of_user
+ "~" + LoginUserAgent.surname_of_user + "~" + StartPage.Course
+ "~" + ExamPage.requestType + "~" + Exam_type);
send(request);
myAgent.addBehaviour(recieveStudentRequestResponse);}};
// behaviour for receiving student request
public Behaviour recieveStudentRequestResponse = new CyclicBehaviour() {
public void action() {
ACLMessage msg = myAgent.receive(StudentRequestResponse);
```

```
if (msg != null) {
message = msg.getContent();
System.out.println(" Agent " + msg.getSender().getName() + " message is " +
message);
messageArray = new String[2];
messageArray = message.split("~");
if (messageArray[0].equals("Attention")) {
ExamPage.examAttentionInfoLabel.setText(messageArray[1]);
ExamPage.examAttentionInfoLabel.setVisible(true);}
if (messageArray[0].equals("Extra time")) {
ExamPage.examExtratimeInfoLabel.setText(messageArray[1]);
ExamPage.examExtratimeInfoLabel.setVisible(true);
myAgent.addBehaviour(extratimeRequest);}} else {block();}}};
// behaviour for sending extratime request
public Behaviour extratimeRequest = new TickerBehaviour(this, 10000) {
protected void onTick() {
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("InfoExtractAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("GetExtratimeRequest");
request.setContent(StartPage.Course + "~" + Exam_type);
System.out.println("Agent " + getLocalName() + ": sending Extra Time REQUEST
message... " + StartPage.Course + "~" + Exam_type);
send(request);
if (addChecker == 0) {
myAgent.addBehaviour(receiveExtratime);
addChecker = 1;}}};
// behaviour for recieving extra time request
public Behaviour receiveExtratime = new CyclicBehaviour() {
 public void action() {
ACLMessage msg = myAgent.receive(extraTime);
if (msg != null) {
myAgent.removeBehaviour(extratimeRequest);
message = msg.getContent();
extra_time = Integer.parseInt(message);
System.out.println(" Agent " + msg.getSender().getName() + " message is Extra
Time : " + message);
extra_timeChecker = 1;
myAgent.removeBehaviour(receiveExtratime);
} else {block();}}};
/* behaviour for sending requests about the statistics of the exam questions
no of questions answered, no of unanswered questions etc*/
public Behaviour sendQuestionsStatistics = new OneShotBehaviour() {
```

```
public void action() {
//System.out.println("Agent " + getLocalName() + ": sending REQUEST
message...");
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("SendNoOfQuestions");
request.setContent(LoginUserAgent.UserId + "~" + StartPage.Course + "~" +
Exam_number);
send(request);
System.out.println("Agent " + getLocalName() + ": sent No of Questions REQUEST
message..." + LoginUserAgent.UserId + "~" + StartPage.Course);
myAgent.addBehaviour(receiveQuestionsStatistics);}};
// behaviour for receiving exam questions statistics
public Behaviour receiveQuestionsStatistics = new CyclicBehaviour() {
public void action() {
ACLMessage msg = myAgent.receive(noOfQuestionsTemplate);
if (msg != null) {
message = msg.getContent();
System.out.println(" Agent " + msg.getSender().getName() + " message is " +
message);
messageArray = new String[4];
messageArray = message.split("~");
ExamPage.QuestAnsLabel.setText(messageArray[2] + " of " + messageArray[3]);
ExamPage.noOfUnansLabel.setText(messageArray[1] + " of " + messageArray[3]);
if (Integer.parseInt(messageArray[2]) > 0) {
DateFormat dtform = new SimpleDateFormat("HH:mm:ss");
Date dtobj = new Date();
String dateStr = dtform.format(dtobj);
System.out.println("Length: " + dateStr.length() + " Hour: " + dateStr.substring(0, 2)
+ " Mınutes: " + dateStr.substring(3, 5) + " Seconds: " + dateStr.substring(6, 8));
int hour = Integer.parseInt(dateStr.substring(0, 2));
int min = Integer.parseInt(dateStr.substring(3, 5));
int seconds = Integer.parseInt(dateStr.substring(6, 8));
double current_Minutes = (hour * 60) + (min) + ((1 / (double) 60) * seconds);
System.out.println("The time in mins: " + current_Minutes);
double minutes_Spent = current_Minutes - initial_Minutes;
estimated_Time = (minutes_Spent / (double) Integer.parseInt(messageArray[2])) *
(double) Integer.parseInt(messageArray[1]);
DecimalFormat df = new DecimalFormat("#.##");
String estimatedStr = df.format(estimated_Time);
ExamPage.estimatedTimeLabel.setText(estimatedStr + " mins");
System.out.println("estimated time " + estimated_Time);
```

```
}} else {block();}}};
// behaviour for sending exam script request
public Behaviour sendScriptRequest = new OneShotBehaviour() {
public void action() {
//System.out.println("Agent " + getLocalName() + ": sending REQUEST
message...");
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
AID aid = new AID("ExamServerAgent@192.168.5.35:1099/JADE", AID.ISGUID);
aid.addAddresses("http://192.168.5.35:7778/acc");
request.addReceiver(aid);
request.setOntology("SendExamScript");
request.setContent(LoginUserAgent.UserId + "~" + Exam_number);
send(request);
System.out.println("Agent " + getLocalName() + ": SCRIPT REQUEST message..."
+ LoginUserAgent.UserId + "~" + Exam_number);
myAgent.addBehaviour(receiveExamScript);}};
// behaviour for receiving exam script request
public Behaviour receiveExamScript = new CyclicBehaviour() {
public void action() {
ACLMessage msg = myAgent.receive(examScriptTemplate);
if (msg != null) {message = msg.getContent();
System.out.println(" Agent " + msg.getSender().getName() + " message is " +
message);
String length = message.substring(0, message.indexOf('~'));
message = message.substring(message.indexOf('~') + 1);
messageArray = new String[Integer.parseInt(length)];
messageArray = message.split("~");
ResultPerformancePage.eScpt.model.setNumRows(0);
for (int i = 4; i < messageArray.length; i = i + 9) {
ResultPerformancePage.eScpt.model.addRow(new Object[]{messageArray[i],
messageArray[i + 1],messageArray[i + 2], messageArray[i + 3], messageArray[i +
4], messageArray[i + 5],messageArray[i + 6], messageArray[i + 7], messageArray[i
+ 8]});
ResultPerformancePage.eScpt.totalQuestionsLabel.setText(messageArray[0]);
ResultPerformancePage.eScpt.questionsAnsweredLabel.setText(messageArray[1]);
ResultPerformancePage.eScpt.correctLabel.setText(messageArray[2]);
ResultPerformancePage.eScpt.wrongLabel.setText(messageArray[3]);}} else
{block();}}};
/*behaviour for keeping track of remaining time, displaying hurry up reminder,
adding extra time to remaining minutes keeping tack of deactivated time
and other time related fuctions */
public Behaviour Timekeeper = new TickerBehaviour(this, 60000) {
protected void onTick() {
DateFormat dtform1 = new SimpleDateFormat("HH:mm:ss");
```

```
Date dtobj1 = new Date();
String dateStr1 = dtform1.format(dtobj1);
System.out.println("Length: " + dateStr1.length() + " Initial Hour: " +
dateStr1.substring(0, 2) + " Minutes: " + dateStr1.substring(3, 5) + " Seconds: " +
dateStr1.substring(6, 8));
int chour = Integer.parseInt(dateStr1.substring(0, 2));
int cmin = Integer.parseInt(dateStr1.substring(3, 5));
int cseconds = Integer.parseInt(dateStr1.substring(6, 8));
if (extra_timeChecker == 1) {
deactivation_Minutes = deactivation_Minutes + extra_time;
.extratimeAddedLabel.setForeground(Color.red);
ExamPage.extratimeAddedLabel.setText("Extra Time (" + extra_time + " mins) has
been added");
ExamPage.extratimeAddedLabel.setVisible(true);
extra_timeChecker = 2;}
current_Minutes = (chour * 60) + (cmin) + ((1 / (double) 60) * cseconds);
double remaining_Minutes = deactivation_Minutes - current_Minutes;
DecimalFormat dff = new DecimalFormat("#");
String remaining_MinutesStr = dff.format(remaining_Minutes);
System.out.println("The time in mins: " + current_Minutes);
Duration = Integer.parseInt(remaining_MinutesStr);
ExamPage.TimeLabel.setText("" + Duration);
if (Duration == halfTime && extra_timeChecker == 0) {
ExamPage.requestExtratime.setEnabled(true);}
if (Duration == (int) halfTime / 2) {
ExamPage.TimeLabel.setForeground(Color.RED);
ExamPage.minsLabel.setForeground(Color.RED);}
if (Duration <= littleTime) {
ExamPage.outOfTimeImage.setVisible(true);
ExamPage.outOfTimeLabel.setVisible(true);}
if (Duration == 0) {
ExamPage.checker = 2;
Exp.close();}}};
protected void setup() {
Exp = new ExamPage(this);
System.out.println("Exam Monitor Agent is ready....");
Exp.setVisible(false);}
protected void takeDown() {
System.out.println("Agent terminating....");}}
```