

# **Multi Agent System for E-Commerce**

By

**Abdullah Alnaffakh**

000807065

*Submitted in Partial Fulfilment of the  
Requirements for the Degree of*

**BEng (Hons) Computer Systems And Software  
Engineering**

**Supervisor:** Radostinka Dontcheva



**Electronic, Electrical and Computer Engineering  
Faculty of Engineering and Science**

06 April 2016

# Abstract

The system introduced within this report provides a user friendly alternative to carrying out e-commerce related tasks, by the implantation of a Multi-Agent system and making use of its capabilities to automate e-commerce tasks as if they were carried out by a user, the system aims to assist buyers and sellers.

A software Agent is a software that is capable of carrying out tasks automatically and independently of a human user. Agents are defined by a number of capabilities such as learning and intelligence. An agent system comprised of a number of Agents that communicate and coordinate their behaviour with each other is referred to as a Multi Agent System (MAS).

A prototype has been implemented successfully demonstrating the beneficial use of a Multi-Agent systems capabilities and proofing its feasibility for implementing e-commerce related solutions. The prototype automates the majority of tasks a buyer or seller may make before buying/selling an item.

Although that a prototype have been implemented, there is always room for future work. The recommended future work has been presented within the validation stage as part of verifying implemented requirements and within the verification stage. Future work includes recommended additional features such as support for additional platform and the use of other agent capabilities.

# Acknowledgements

Thanks to the project supervisor for her continues support throughout the development of the system and her consistent and quick reply to any related queries. Without her support it would have been much more challenging to assess whether the project is heading in the correct path or not.

The advice provided by the second marker is much appreciated as it assisted in fine tuning this report.

The resources made available by the IET and Cambridge University played a huge role in providing the details required to study and research the Software Agents and Multi-Agent fields, especially the Knowledge Engineering Review journal, without the availability of this information it would have been much more challenging to gain as much information as gained now.

## Contents

<b>CHAPTER 1 INTRODUCTION .....</b>	<b>6</b>
OVERVIEW.....	6
1.2 AIMS AND OBJECTIVES .....	7
1.3 MAIN DELIVERABLES.....	8
1.4 LITERATURE SURVEY.....	8
1.4.1 Background.....	8
1.4.2 Multi Agent Systems in E-commerce.....	9
1.4.3 Multi Agent Systems and User Driven Applications .....	12
1.4.4 Capabilities of Multi Agent Systems.....	13
1.4.5 Issues and problems in Multi Agent Systems.....	14
1.4.6 Literature review summary .....	15
<b>CHAPTER 2 ANALYSIS.....</b>	<b>17</b>
2.1 INTRODUCTION.....	17
2.2 POTENTIAL SOLUTIONS.....	17
2.2.1 Solution 1 .....	17
2.2.2 Solution 2 .....	18
2.2.3 Solution 3 .....	19
2.3 CHOSEN SOLUTION .....	19
2.4 PROJECT REQUIREMENTS.....	20
2.4.1 Requirements Elicitation.....	20
2.4.2 Non-Functional Requirements .....	21
2.4.3 Functional Requirements.....	22
2.4.4 Resource Requirements.....	24
2.5 FEASIBILITY ASSESSMENT .....	25
2.5.1 Solution feasibility.....	25
2.5.2 Technology feasibility .....	25
2.5.3 Technical Feasibility.....	26
2.5.4 Schedule feasibility.....	27
2.5.5 Economic Feasibility.....	27
2.6 RISK ASSESSMENT .....	27
2.7 DEVELOPMENT TOOLS AND LANGUAGES .....	28
2.7.1 Development tools and software.....	28
2.7.2 Programming Technologies.....	31
2.7.3 Methodologies .....	34
2.7.4 Choices Made .....	36
2.8 ANALYSIS (O-MASE) .....	37
2.8.1 Problem analysis .....	37
2.8.2 Solution analysis .....	41
2.7 PRELIMINARY DESIGN .....	42

<b>CHAPTER 3 DESIGN .....</b>	<b>44</b>
3.1 DATABASE DESIGN .....	44
3.2 GRAPHICAL USER INTERFACE (GUI).....	45
3.3 ARCHITECTURE DESIGN (O-MASE) .....	47
3.3.1 Model protocols.....	47
3.4 LOW LEVEL DESIGN (O-MASE) .....	48
3.4.1 Plan model .....	48
3.4.2 Actions model.....	55
3.5 CLASS MODEL .....	59
3.6 AGENTS ASPECTS .....	60
<b>CHAPTER 4 IMPLEMENTATION.....</b>	<b>63</b>
4.1 DATABASE IMPLEMENTATION.....	63
4.2 CLASS MODEL IMPLEMENTATION.....	64
4.2.1 The database interactions class .....	64
4.2.2 UI Class implementation (GUI).....	67
4.2.3 The CommClass (Communications) .....	73
4.2.4 The message exchange class .....	77
4.2.5 The Agents Class.....	79
4.2.6 The SellingAgent Class .....	80
4.2.7 The BuyingAgent class.....	81
4.2.8 AgentSys class (main).....	83
<b>CHAPTER 5 TESTING .....</b>	<b>85</b>
5.1 WHITE BOX TESTING .....	85
5.2 BLACK BOX TESTING .....	95
<b>CHAPTER 6 VERIFICATION.....</b>	<b>96</b>
<b>CHAPTER 7 VALIDATION .....</b>	<b>98</b>
<b>CHAPTER 8 CONCLUSION.....</b>	<b>101</b>
<b>CHAPTER 9 REFERENCES .....</b>	<b>104</b>
<b>CHAPTER 10 APPENDIX .....</b>	<b>107</b>
GANTT CHART.....	107
SYSTEM CODE.....	109
Ulclass.JAVA .....	109
DBinteractions.JAVA .....	113
CommClass.JAVA.....	114
MessageExchange.JAVA .....	115
Agents.JAVA.....	116
TheBuyingAgent.JAVA.....	117
TheSellingAgent.JAVA .....	118
AgentSys(main).JAVA.....	118
XML COMMUNICATION FILES.....	119

# Chapter 1 Introduction

## Overview

Most business that exist today are providing their consumers with alternative ways to shop, for example Argos has many branches that are distributed across the nation but they also host a website which is linked to a database containing all the products they sell, making the same options that are available instore also available online through their website, this is the case with many business.

The amount of consumers which make online purchases is increasing on yearly basis due to numerous reasons such as finding it more significant to browse for and purchase items online rather than having to visit many high-street stores (FFA, 2015).

As the number of online shoppers increase so does the number of shopping sites, and online based business, such as Amazon.

This has led to the availability of an enormous quantity of retail sites a user can access to browse for their desired items, although that most business aim towards making their sites as much user friendly as possible but the process of going through many sites is not user friendly, especially to users who have just started using computer system.

Distributed systems such as the internet can be quite complicated and time consuming to interact with, for example a user which is willing to purchase a particular item online may need to browse through to available sites which sell that item and carry out a number of tasks as: finding a sites that advertise products in the same currency as the users currency and assuring that the seller must be able to deliver purchases to the users country and communicate using the same language as the user, this has introduced a problem which is the amount of time consumed by each user to manually undertake an increasing number of tasks and considerations to purchase a product online.

Later on within this report further problems and tasks an online shopper goes through will be identified and then a feasible solution will be proposed to tackle those issues and problems.

The system presented within this report is aimed towards making e-commerce a more user friendly experience for buyers as well as sellers.

The system will be developed for a client that specialises in the development and retail of software e-commerce systems.

The problem will be tackled through the use of an engineering methodology which would assist in progressing through the development of the system coherently, the engineering methodology will be chosen within the analysis stage where a number of methodologies will be compared in order to choose the most suitable one.

No information will be provided within this section of the document detailing the technologies used to develop such system as much details will be provided within the feasibility study justifying why such a technology has been used as it isn't professional to blindly choose a technology without assessing its feasibility.

## 1.2 Aims and Objectives

The main aim of the project is to develop a system that creates a user friendly way of making online purchases and sales by automatically carrying out the tasks a user would normally undertake to make an online purchase, and making decisions on their behalf, such tasks include:

- Finding the product at a reasonable price based on the current average price that product is being sold at currently and recommend the site to the user of where the product is being advertised.
- Assist the user in choosing a reputable sites to purchase items from to avoid fraud and phishing sites.

More of these tasks would be included within Feasibility study of this report where the proposed solution will be presented, further functionalities will be included in the project requirements document as information will be gathered from different sources (end users, organisation) to identify the project requirements and functionality.

By carrying out the above on behalf of the user the user would not have to browse individual sites with many tabs open at the same time and compare the information on each site to find a suitable preference, it will also help to reduce the risk of falling as a victim for online fraud.

The objectives of the system are defined below:

1. Automate e-commerce related tasks, such as searching for retail sites with a good rating that is based on user reviews.
2. Efficiently use the properties and features of agent systems to achieve the systems aims.
3. The functionalities of the system should be based upon tackling the negative aspects of the current manual process.
4. The system must be as user friendly as possible in every aspect, as this is part of the systems aim.
5. Assessing whether the project is feasible or not, as well as identifying potential risk that may rise during the life time of the system or within its development
6. Choosing a particular methodology that would be most suitable to implement the proposed system with.
7. Analysing current development tools and programming technologies to define which ones would be most suitable to develop the system with.
8. Verifying and validating whether the developed system contained the defined functionalities in order to assure the system has achieved its goal.

## 1.3 Main deliverables

As a result of completing the development of the system a number of outcomes will be expected, such outcomes are included below:

- A prototype that contains the main functionalities which have originated from the projects requirement document. The prototype would be a much smaller version of the system as the number of functionalities will differ as some have a priority over others, for example: communication between agents is more important than a fancy user interface.
- Project documentation: highlighting every stage throughout the development of the system, this may include the initial documents generated prior to the development of the system such as the feasibility study.
- Project files including source code: files generated as part of the development of the system will be made available, therefore if further updates are to take place in the future for the system to be more efficient then the project files are already available.
- An installation file which allows the installation of the prototype.
- User Guide: To provide a guideline of how to use the system explaining the role of different components within the user interface, such as text fields.
- A testing document highlighting system tests undertaken as well the outcomes of such tests.

Table 1.1 will include more information that is associated with each one of the deliverables:

**Table 1.1 Deliverables Table**

<b>Deliverable</b>	<b>Structure</b>	<b>Approved By</b>	<b>Resources</b>
System Prototype	Files and Databases	Project supervisor, Client	Development tools and software, Workstations
System Tests	Document	Project Manager	Word Processor.
User Guide	Document	Client & Project Manager	Word Processor.
Final Report including all project documentation	Documents	Project Supervisor, Client	Client, Project Manager, Word Processor

## 1.4 Literature Survey

### 1.4.1 Background

The technology that has been chosen to develop the system with is Multi Agent systems, therefore the literature survey will be based on this technology, it has not been justified yet



why this technology is deemed suitable as this information will be within the feasibility study where there will be multiple justification as to why this technology is deemed feasible to tackle such issues.

Multi agent systems are software programs which are capable of carrying out autonomous behaviour as well as having many capabilities such as learning and decision making, an agent system which consist of more than one agent is referred to as a Multi Agent System.

The literature survey carried out will contain a number of themes where each theme relates directly to the chosen technology, the relativeness of each theme will be summarised as well as assessing the strengths and weakness of related work and research that has been done.

Certain themes of research have been included below, each theme reviews of a number of literature which relate to the theme.

Information which has been gathered from this survey will assist in making decision throughout different stages of the system development, for example one of the themes is comparing the implementation of normal user driven applications versus the implementation of an autonomous Agent system. Such a theme will assist in choosing whether a user driven application is more suitable to be implemented or the implementation of an MAS.

Other themes included such as the theme titled *Issues and problems with Agent* system will assist in carrying out the development of the system while attempting to avoid issues which others have faced or to use the same solution used if such problem is faced, for example in the theme titled *Issues and problems with Agent*, one author mentions that the development tools available are not friendly and efficient while other authors recommend a particular tool that avoids this issue.

#### **1.4.2 Multi Agent Systems in E-commerce**

The Use of Multi Agent Systems in E-commerce is increasing, some of these systems have a purpose similar to the one proposed in the feasibility study and other are there to provide an alternative method towards manual transaction processing, some of these systems are reviewed below to identify technologies and methodologies used and to assess the similarities between the proposed system and those which have already been developed:

(Zeng, 2009) Has developed a Multi Agent system to assist in online shopping similar to the proposed system within this report but there are a number of differences between the functionality, agents, and objective of the system.

The main aim of the system developed by Zeng is to assist in recommending a suitable product to the user (shopper) based on their preference by searching through websites, the number of agents within the system will total five agents where each agent has a particular objective, the agents are: an interface agent, a buyer agent, an expert agent, an evaluation agent, and a collaboration agent, only a few agents will be discussed below:

- The interface agent: its purpose is to gather information from the user about the product they desire as well generating technical questions that assist the agent in its search, for

example if the user wishes to purchase a PC the agent will generate questions such as what would be the ideal HDD capacity, then it will pass the collected information to the buyer agent.

- The buyer agent: once information about the product is retrieved, the buyer agent will search the internet for an advertiser which advertises the same product and its specifications then it will consult the user if they would like to proceed, if so then the agent will contact the seller to request a deal or offer.
- Expert agent: this agent will embed expert's advice and opinions about products to assist in the decision making of which product to suggest.

The system will theoretically achieve its aim but there are a number of points which raise concerns, such points are:

- The developer uses five agents where each agent has a particular objective although this is helpful in simplifying the objective of the system by breaking down the functionalities into smaller chunks but the use of certain agents can be avoided to improve performance, for example the buyer agent can also collect information about the product rather than having the interface agent carry out such task.
- It is mentioned that the interface agent will gather technical information about the product a user wishes to purchase but what if the user is not familiar with the product they wish to purchase, would any help be provided to assist the user in choosing the technical specifications about their product?

The developer specifies the development tool used (JADE) and also defines the method used for decision making which is the multi attribute decision making method (Barbuceanu et L, 2000).

There are a number of differences between the system developed by Zeng, and the system presented within the feasibility section of this report, such differences are:

- The number of agents is two as opposed to five.
- Although the objective is similar but the functionalities of the agents differs, for example the author does not take into consideration any security risks or gives the user configuration privileges over the behaviour of each agent.
- The system developed by Zeng is aimed to assist buyers only while the system presented within the feasibility study is aimed towards assisting buyers as well as sellers.

(Helmy, 2007) Presented a framework for collaboration between Multi Agent systems in E-commerce in the aim of assisting consumers in carrying out online purchases which suit their preferences and interests.

Where an agent system would consist of three agents, a mall agent, a buyer agent, and a seller agent, each briefly introduced below:

- The buyer agent: will gather information about the user preferences in order to compare them with products available online.

- The seller agent: the seller agent gathers information about advertised products online and their sellers then it stores them within a data base where product information would be available upon other agent's requests.
- The mall agent: handles and sends queries from/to both agents, once a buyer agent has collected information about the user preference it will communicate this information to the mall agent where the mall agent will search for a matching product based on the information provided to it from the seller agent.

Each agent within the system will have its own local database that is only accessible by the agent itself, it will store information that is relevant to the agents objective, for example the seller agent holds information about the seller of each product, the agent will pass information from one database to another agent as part of the queries between agents which will be in the form of XML messages.

The author mentions that the system has been developed and tested, and specifies the technologies and development tools used such as XML, TCP (As message exchange is through a network), and JADE as the development software.

The author has failed to mention certain aspects which makes part of the systems functionality unclear for example the author does not mention how the seller agent will retrieve product information, is it from an internet resource or do different sellers have agents that passes out product information about products advertised on their sites.

There are a number of similarities and differences between Helmys' system and the one presented in the feasibility study:

- The number of agents, two agents are used where communication is direct in between those agent rather than having a middle agent such as the one presented in Helmys' system.
- The agents within the system proposed in the feasibility study will communicate locally as both agents are stored on the same workstation, while in Helmys' system communication will take place over the network.

The use of multi agent system in e-commerce is not limited to transaction processing but also to different aspects such as the objectives of the systems mentioned below:

(Collins, Ketter and Gini, 2002) developed a Multi Agent system called MAGNET which compares products from many sellers on behalf of users based on different attributes then presents the comparison to a user.

(Wei Moreau and Jennings. 2003) Developed a Multi Agent system where each agent contributes in an auction then another agent will assess each auction presented by each agent to determine the most profitable offer after doing so it will consult the user if they wish to proceed then that agent will proceed with making a bid.

Agents are widely used in the gaming industry where each game character is represented by an Agent.

### 1.4.3 Multi Agent Systems and User Driven Applications

When a problem is presented to a team of engineers normally the issue can be tackled through a number of potential solutions where the most feasible solution would be chosen based on different consideration such as ease of deployment and sustainability.

Similarly there might be scenarios where the use of an normal desktop application that is user driven may be favoured as a solution more than the use of Multi Agent systems, as each system has different capabilities and its own properties, for example desktop application with friendly user interfaces allows users to input data and view an output as a result of an algorithm being executed, while in multi-agent systems agents are able to interact with each other for similar outcomes without user instruction because of their autonomous property (Aylet et al, 1998).

For example: the development of a tilling system would not benefit from the capabilities of an agent system such as AI (Artificial intelligence). A multi-robot system that requires coordination and decision making requires the use of Multi Agent systems, as Multi Agent systems are capable of coordinating tasks and making decisions, such multi-robot systems that are currently used in the real world are Sony Legged robot league (A team of robots capable of playing football), Other uses of multi agent-systems can be in service robots (Weiss, 1999).

The uses of Multi Agent systems technologies in systems where the capabilities of agents can be used, has also been recommended by Minsky (a researcher in the field of AI) in his book *Society of Mind Paradigm* (Minsky's, 1986).

In their paper (Aylet et al, 1998) on agent systems and applications, the authors also mentions that the use of multi agent systems is different than those of normal application emphasising on the fact that agent system are capable of decision making as part of their autonomous behaviour as oppose to application that may require direct input from a user.

The author provides many examples of where Agent systems are used, one of the uses was in air traffic control systems and another was in automated flights (Weiss, 1999).

Authors which have recommended the use of Multi Agent systems over applications have focused on characteristics of multi-agent systems mainly as a bases for comparison with normal application while agents can have many characteristic and not necessarily all of them are used within the development and deployment, there can also be other ways to compare normal application with agents such the availability and reliability of development tools.

There are system where agents and user driven applications can interact with each other but the authors fails to mention such scenario an example of this is the agent Clippie in Microsoft office 2003, where the agent will continuously try to offer help depending on what the user is doing.

The information within this theme will assist in choosing whether a user driven application is more suitable than Multi Agent systems.

#### 1.4.4 Capabilities of Multi Agent Systems

This theme will elaborate on some of the capabilities of Multi Agent Systems, in order to assist in choosing whether such capabilities should be used within the proposed system or not, each capability will be briefly summarised based on authors description of these capabilities as well including examples of where such capabilities can be used.

One of the reasons why the multi-agent system field is referred to as multidisciplinary is due to the fact that each capability of an agent relates to a field of its own such as AI and Robotics.

**Negotiation:** is one of the forms which agents may use to communicate with each other in order to achieve a particular goal or complete a certain functionality in the interest of both agents, for example two agents could be negotiating with each other within an auction (Beer et al, 1999).

Negotiation does not necessarily have to be in one form or relating to a particular context, negotiation can take many forms such as argumentation and bargaining.

The developers of negotiation protocols normally set a number of rules, some of these rules are identifying the eligible participants and making a decision to change the negotiation status when an agreement is reached.

The decision made by a particular agent is not independent from other agents or machines rather the decision is based on communications made between agents (Aylett et al, 1998), for example an agent system which calculates the time to travel through a certain route will base its calculation on information has received from a traffic monitoring agent.

Both authors above have failed to mention how would one agent be aware of the communication protocols of other, for example what if one agent sends a message to another agent within a third party system but the receiving agent has failed to interpret that message and reply to it because their communication protocols are different, however (Dastani, Jorge and Gomez-Sanz, 2005), does mention that there is very little focus in regards the implementation of such capabilities.

It is not mentioned that a particular technology is used to establish communication between agents, but it has been mentioned by one author previously (Helmys', 2007) where messages where generated in XML and transferred through the use of TCP across a network, others did not specify how messages where exchanged between agents indicating that there is not a defined protocol or standard and technology used for message exchange.

Both authors have agreed that negotiation between systems plays an important role in decision making.

**Learning:** Throughout the design stage of multi agents system it would not be possible to predict every single encounter an agent may come across (Alonso et al., 2001), and it will be insufficient to repetitively integrate encounter handling throughout the lifetime of a multi agent system as that may mean that the system is in a never ending development cycle, if a learning algorithm is implemented in such system then encounters that an agent may face would not result in such worry as the agent will learn how to react in certain cases.

As research proves that the learning ability is one of the most important features in Multi Agent Systems (Russell & Norvig, 1995; Huhns & Singh 1998), learning contributes towards how efficient the decisions made by an agent and its level of intelligence.

There is a number of techniques in which agent learning can be implemented with for example social learning is one technique where agents within one environment passes its knowledge to new agents that have just entered this environment.

There can also be other learning techniques, such as single agent learning and multi agent learning.

Single agent learning is concerned with the improvement of agents own skill (Stone and Veloso 1998) as oppose to the entire multi agent system, single learning aspects do not necessarily mean that they will not be sufficient in multi agent system as it would still be possible to achieve coordinated behaviour (Sugawara & Lesser, 1993).

It has been proven that in certain cases multi agent learning has decreased performance of the multi agent system (Mundhe & Sen, 2000), in such cases it may be better to use single learning techniques.

**Other capabilities may include:** There are many other capabilities of Multi Agent systems which may include, coordination, planning, decision making, information sharing, these capabilities will not be expanded on.

This theme has assisted in choosing the capabilities which can be used within the proposed system, these capabilities have been included within the feasibility study with a justification to why each capability is deemed useful.

### **1.4.5 Issues and problems in Multi Agent Systems**

Before undertaking the development of a system it is beneficial to become familiar with any known issues related to the technology that will be used to develop the system, being familiar with such issues will help to avoid encountering that issue or to use a solution someone has already used or suggested previously if they have encountered the same issue.

The fact that multi-agent systems is a multidisciplinary field means that issues that are commonly encountered with such system relate to issues with one of the fields which an agent relates to, for example unsolved issues with the AI field may inherently lead to issues if the AI capability is to be implemented within an agent system, another example would be the implementation of agents within a robotic system may have issues that are related to physics and mechanics. (Luck, 1997) also (Dastani, Jorge and Gomez-Sanz, 2005).

There are also other issues that relate to the development and implementation of such systems such as the programming languages used which practitioners claim that they do not ease the implementation of Agents and they often suggests the development of a new programming technologies that is specially designed for the purpose of implementing Agent systems, practitioners also complained that methodologies emphasise on the design rather than implementation (Dastani, Jorge and Gomez-Sanz, 2005, Luck, 1997, Aylett et al, 1998).

(Aylet et al, 1998) categorises the issue within the field into two main categories and refers to them as two barriers which are preventing the uptake of Multi Agent Systems, the first barrier she has recognised as a social barriers where the industry lacks of professionals who specialise in such field or who have studied it as part of their degrees. And the other barrier is technical relating to the development tools and languages.

(Dastani, Jorge and Gomez-Sanz, 2005) as well as agreeing that there is a technical barrier and suggests new programming languages and development tools to be developed in order to ease the development of multi-agent application, the author also suggest that the methodologies currently in use emphasises on the analysis and design rather than the implementation.

A good point has been raised by the author (Aylet et al, 1998) and (Dastani, Jorge and Gomez-Sanz, 2005) is the lack of graduates which have studied or specialised in Multi Agent systems, if the number of specialist in the field increases then the possibility of resolving current issues will be increased as well, although that practitioners and academics regularly meet through workshops and seminars where issues are discussed, it does not seem like a solution is suggested for this particular issue, it could be as simple as introducing Agent Systems into the curriculum of related degrees to grasp the attention of students.

All authors which have spoken about these issues encouraged and supported online discussion forms, workshops and collaborative research between academia and professionals to help resolve some of the identified issues.

This theme has assisted in identifying the areas which may be of concern or require attention where special attention will be given when choosing the methodology, development tools and programming technology, where each will be compared based on what practitioners recommend.

#### **1.4.6 Literature review summary**

As mentioned within the introduction of the literature survey, each theme has a defined purpose, the information gathered from each theme will assist in the decision making stages throughout the

The first theme dealt with reviewing the current MAS systems involved in an e-commerce scenario, this theme has helped in providing an insight to what technologies and methodologies were used to develop such system. The critical review of systems within this theme will assist in avoiding the implementation of features which may not be so beneficial. The review of these systems assisted in identifying how the system proposed within this report stands out from other systems that may have the same purpose.

The second and third theme assisted in determining whether the implementation of an MAS is more feasible than the implementation of a user driven application and why, as a result an MAS system was chosen over a user driven application, as the capabilities of agents can be made use of to achieve the aims and objectives of the system. Further justification for such choice is included within the feasibility study.

The fourth theme assisted in identifying the challenges currently faced within this field, as such challenges have an impact on the development of the system. Therefore this theme has provided

an insight to what issues may arise during the development of the system and how to tackle them.



# Chapter 2 Analysis

## 2.1 Introduction

The following is a brief summary of the problem (previously introduced in the overview):

When a user is willing to use e-commerce whether it was for selling or purchasing products normally they have a significant amount of considerations to make especially with the increasing amount of retail sites, the number of considerations a user goes through can be time consuming and may repulse some users from carrying out online purchases as they may consider the process to be complicated and unfriendly, some of these considerations are:

- Assuring that the retailer is able to deliver to the user's country.
- Comparing the price of the same product from many sites, in order to assure that the advertised product is not overpriced on some websites.
- Searching for sites that sell the product a user desire.
- Assuring the retailer is a trusted retailer and not a phishing site.
- Assuring that the retailer is reputable based on consumers reviews.
- Some consumers view their products on daily basis and keep an eye out for any offer.

In order to reduce the amount of consideration or tasks, some users prefer to continuously use one site but not all websites sell every product a user may desire, For example Amazon has a wide range of products but that's not the case for car parts, therefore a user would have to eventually refer to different sites.

The ideal solution would be a system that creates a user friendly method of interaction with e-commerce, the system should have autonomous capabilities that carries out tasks on behalf of users such as browsing the internet for deals and other related tasks such as the ones mentioned above.

## 2.2 Potential Solutions

### 2.2.1 Solution 1

A Multi Agent system that will consist mainly of two agents, each assigned with a set of functionality in order to achieve a particular objective, both agents will be involved in an E-commerce scenario where one agent will assist users in purchases and another will assist them with selling items, the multi agent system is independent of a particular website or organisation neither it would be limited to particular items or products.

**The role of the buying agent:** The buying agent will search the internet for an item the user is looking for, rather than the user having to browse individual websites on the internet and assuring that related item aspects suites the user's needs, such aspect would be currency used

to purchase the item and suitable price, the agent should be able to carry out tasks and considerations on behalf a user when they wish to purchase an item.

**The role of the selling agent:** The selling agent will provide the user with information of how much his/her item can be sold online whether it is a second hand item or brand new, information which is feedback to the user would include lowest/highest price the item is being sold at, as well as which website it is being sold at with its rating if available, as advertising products on reputable site will increase the chance of selling the advertised item.

Each agent will be visually available for the user as an animated character for entertainment purposes, the animated character has not been drawn yet but it would most likely be a simple 2D character.

Both agents will communicate with each other to provide the user with more useful information, for example when a user purchases an item the buying agent will communicate this to the selling agent therefore the selling agent will notify the user that if they want to sell this item in the future it can at least be sold at this price.

Communication between agents may take the form of message exchange between the two agent as a well as in an animated scenario in the purpose of entertaining the user, for example the agents may occasionally play fight with each other, or pretend that they are carrying out important research while gathering information of the internet or taking a nap every now and then.

The buying agent would include a wish list where the user would be able to include a list of items they wish to purchase, the agent will use this information to monitor retail websites daily for offers, therefore if the items price becomes cheaper than usual the agent will notify the user of this.

The user will have the option to limit the agent's interactions to particular sites rather than the agent browsing through all available sites which meet the search criteria, for example search for the price of this item only on this particular site rather than a global search.

The brief summary of the system introduced above contains a set of functionalities which are derived on what ideally is required for the system to be desirable by users as well as friendly, as user interaction would be limited, (what is meant by limited is that there will not be many configuration a user will have to go through an understand in order to assure that the system will function correctly, after all it is autonomous).

Further functionality of the system will be defined within the projects requirements document after information has been gathered from those which the system relates for example users, the organisation and technical experts within the organisation.

## **2.2.2 Solution 2**

A Multi agent system that carries out the functionalities of solution 1 but in addition rather than recommending sites to the user it would be able to carry out purchases on behalf of the user by memorising the users personal information as well as payment details which would also be a user friendly way to interact with e-commerce as the user will not have to figure out how the

site works to make a purchase rather it would be the agents responsibility but if the agent system carries out purchases which do not interest the user then most users will most likely not trust the decision making capability of the system.

### 2.2.3 Solution 3

Another solution would be the use of an agent system to identify and explain different components within e-commerce sites to the user such as a shopping karts or the purpose of a particular button.

But the development of such system will require much more efforts in comparison to solution 1, as it would require the agent to automatically interpret web pages and generate a brief description of the components purpose rather than the description being predefined within the system.

## 2.3 Chosen Solution

One of the potential solutions will be chosen and developed, the choice will be made upon a number of comparable aspects:

**Complexity of implementation in relation to meeting time bounds:** This method of comparison has been chosen to assess the complexity of the solution to be implemented, as the more complex the solution is then the more time it may consume to implement, which leads to the possibility of not meeting the milestones of the project:

1. Solution (1): the main complex feature of this solution would be the identification of a product and related information within a web page then the retrieval of such information.
2. Solution (2): as solution 2 carries out purchases on behalf of the user then it may be complex to identify the fields and components the agent may interact with, as all web pages are displayed differently without consistency, such fields may be credit card number and card holders name, the agent will have to figure out which information goes where.
3. Solution (3): it is unclear how the agent may interpret the contents of a particular site and identify its contents then generate a description for the user, the implementation of such functionalities is not impossible but it is complex, for example the agent may refer to the language which the web page has been developed with to identify UI components.

**Reliability:** Although any solution to be implemented will be aimed to be as reliable as possible however it is easily predictable that any implemented solution may not reach perfection, but some solutions may be more reliable than others.

1. Solution (1): there will always be a possibility where the agent may fail to identify the contents of a particular site, or miss out a site completely but this will not be frequent.
2. Solution (2): some web pages go through security authentication methods when payments are to be made such as displaying the banks secure authentication window

which displays a number of security questions that only the user knows the answers to those questions, the agent will not be able to carry out such task.

3. Solution (3): this could not be determined at the moment as it is unclear how the agent will function, a thorough testing plan will be required to assure the system is functioning correctly.

**User satisfaction:** each solution aims to achieve the systems aim in different ways, however some solutions do so in a more user satisfactory way than others, for example solution 3 indicates that the user has to run several applications at once, this may be unfriendly to some users as they may find the process complicated.

1. Solution (1): there has been a number of consideration made to assure user satisfactory, some of which are included within the risk assessment and other are introduced as part of the solution such as animated characters to entertain users.
2. Solution (2): if the agent did not interpret the web page correctly and identified its contents then user may not be satisfied as the system would not have completed the advertised functionalities and did not fulfil its aim.
3. Solution (3): this will be another system that works along with the web browser, depending on the user's preference they may not prefer to use two different programs concurrently, therefore this may be unsatisfactory for some users.

Based on the comparisons made above, solution 1 is least complex to implement and most reliable as well as the expectancy of being more user satisfactory out of the three solutions, therefore it will be the chosen solution to be implemented, further assessments will be made within the feasibility study to assess the feasibility of the solution.

## 2.4 Project Requirements

### 2.4.1 Requirements Elicitation

The different requirement of the project can be derived from meetings held with those who may interact with the system such as end users and technical experts within the organisation, in those meeting a number of questions have been asked in order to draw out the requirements of the system, these are summarized below:

#### **Consultations with End Users:**

1. What are the disadvantages of carrying online purchases using the current process?
  - Time consuming as many comparisons have to be made between different advertisers to determine the suitable product value and assuring the product to be purchased is not overpriced by the seller.
  - Complicated as when users want to carry out a purchase they do not normally know where to start looking and how to compare different sellers.
2. Do you dislike anything about current software that relates to e-commerce?

- Some software promote certain sellers because of a contract made between the software developers and the advertiser rather than basing promotion on customer needs.
3. In general what do you dislike about the software that is available for use today?
    - The number of notifications generated from applications is enormous which distracts users when they are carrying out unrelated tasks.
    - Lack of help or instructions when needed.
    - The number of configuration a user's goes through is confusing.
    - Loading times can be exceedingly long sometimes.

#### **Consultations with the Organisation:**

1. How will you be making the software available to users?
  - Through a website hosted by the organisation where the system will be included within in the form of a downloadable executable file.
2. Would you like the development of the system to be according a certain standard or using a particular methodology?
  - No particular standard or methodology chosen by the company.
3. Who will relate to the system in the organisation?
  - A team of technical experts which will be in charge of customers' technical support.
  - A team of software engineers who will maintain the system and produce updates when necessary.

#### **Consultation with the team of software engineers:**

1. What would you require in order for you to maintain the software in the future?
  - Availability of source code with the correct use of indents, white space and comments throughout.
  - Project documentation.

### **2.4.2 Non-Functional Requirements**

The requirement elicitation has resulted in identifying some non-functional requirements which are mainly certain constraints that must be applied to the system

- The time it takes to gather data from different sites should be less than 60 seconds as some users become impatient if the loading time within software is consuming too much time.
- There should not be too many notifications generated by the system which may bother users, in addition users must be given the option to disable these notifications.

- As each animated character will carry out some sort of behaviour to entertain the user such as napping for example, there shall not be too many behaviours that distract the user from their main objective.
- The amount of information outputted from the system must be presented in an easily understood manner to avoid any complexity that may dissatisfy users.
- The system should not have demanding hardware requirements to increase the number of compatible users as not all users have machines at the same hardware specifications.
- As the agents will be represented by an animated character each, these agents must not be always displayed.
- The failure rate must be within acceptable bounds, it is predicted that the agents may occasionally not interpret certain sites or their contents but there shall be measures taken to reduce the failure rate.

### **2.4.3 Functional Requirements**

The following set of functionalities identified below will be required in order to be implemented within the system some of these requirements are derived from the consultations made with those who relate to the system and other requirements are there to implement the functionalities of the system:

#### **Functionalities required for the buying agent**

- The agent must be able to search for websites that advertise the product a user is interested in, in addition it must be able to identify a retail site from a non-retail site.
- The agent must be able to identify product related information on the advertisers site, this includes: currency, delivery availability, and stock status.
- The agent must be able to store information collected in order to compare this information later.
- The agent must be able to rate a certain seller based on buyers reviews and review site.
- The agent must be able to compare information collected from different website in order to choose one that is suitable for recommendation to the user, comparisons will mainly be based on price and sellers rating.
- The agent must be able to present recommendation to the user in a friendly way.
- The buying agent must be associated with a wish list.
- The buying agent must be able monitor any changes made in relation to product contained within a wish list and notify the user of those changes, for example if one of the product in the wish list became cheaper than the agent will notify the user of such change.
- A wish list should be associated with this agent, where it holds the items that a user wishes to get daily price updates on.

#### **Functionalities required for the selling agent**

- The agent must be able to search for websites that advertise the product a user is interested in, in addition it must be able to identify a retail site from a non-retail site.

- Identify product related information, includes price and condition (new or used).
- Store information collected by the agent, this information would be the retailers website and price of the item.
- The agent must be able to calculate the average price in which a product is advertised in.
- The agent must be able to rate websites based on buyers reviews and review sites.
- The agent must be able to recommend the user a price which they can sell their product with and recommend a site which their product can be advertised in, this recommendation should be on the sites rating.

There may be some common functionalities between the two agents in such event the same algorithm will be used rather than developing two different algorithms for that same functionality, for example both agents need to assess the rating of a site before recommending it to the user.

### **Communication Requirements**

- A communication protocol must be created in which both agents can comply with.
- Communication will be in two forms, animated and message exchange.
- The buying agent should communicate with the seller agent each time a user carries out a purchase in order for the selling agent to provide the user with information if they do wish to sell the product they have just purchased.

### **Security Requirements**

- As mentioned in the requirements of the selling and buying agent, both agents must be able to rate the site before they recommend it to the user to avoid recommending a fraudulent websites.

### **User Requirements**

- User Require a dedicated field where they can enter product information, e.g. product name or model.
- Hints must be available in the event help is needed.
- Accessibility options must be available in consideration to those who may be visually impaired.
- A friendly user interface.
- A number of configuration which a user can interact with.

### **User Configurations Required**

4. Turn off animation
5. Turn off notifications
6. Limit the agent interaction with one site defined by the user.
7. Limit the agent interaction to multiple sites also defined by the user.

## User Interface Requirements

- An animated character to represent the selling agent and another for the buying agent.
- The number of animated behaviour carried out by each agent must be defined, behaviours may include:
  1. Nap
  2. Play fight
  3. Pretend to carry out research.
  4. Whisper to each other when they are exchanging messages
- Information outputted from each agent must be displayed in a friendly way (as in not randomly displayed in popups or taking over the whole screen).
- A configurations menu where user can change the systems settings.

### 2.4.4 Resource Requirements

There are a number of resources required in order to implement the system, these resources are organised into table 2.1, Some required resources are already available therefore there is no need to purchase them:

**Table 2.1 Resources Required**

Resource	Resource Name	Purpose	Quantity	Availability	cost
Development Workstation	Desktop Computer	To carry out implementation and testing of the system as well as other task	1	Already Available	£0
Image Processing Software	Photo Shop CS6	To generate an animated character to represent agents	1	Already Available	£0
Development Software	Eclipse (MARS)	To implement and test the system.	1	Available for free	£0
Agent Development tool	AgentTool3	To design and implement agents.	1	Available for free	£0

The development tools and software chosen to implement the system have not been chosen randomly rather an analysis (details provided within the analysis stage) has been made on the current available tools and then the most suitable tools and software were chosen to implement this system with.



## **2.5 Feasibility Assessment**

### **2.5.1 Solution feasibility**

The main aim of the ideal solution is to provide a user friendly method of interacting with E-commerce, which eliminates the negative aspects of the current E-commerce interactions. Therefore a number of potential solutions were proposed, and then one of these solutions was selected for development as result of a brief comparison of these solutions.

In order to tackle the problem directly a number of functionalities were included within the chosen solution, each one of these functionalities directly tackles each negative aspect within the current process, for example the current process contained a high number of manual tasks a user undertakes to purchase a product online, based on this, the autonomous capability of agents will be used to perform tasks on behalf of the user.

There are also a number of functionalities which has been included within the system to even enhance the users experience even further, one of the added functionalities is the animated interaction between agents and how each agent is represented by an animated character, in the purpose of entertaining the user.

To assure that the system is feasible from all angles, there has been a number of considerations made in relation to security and user satisfaction, as a result of doing so some functionalities have also been added to the system.

For example it has been looked into what users may find irritating about software. Users were commonly unsatisfied by the number of notification they may receive from a single software, one functionality that has derived from this is including a configuration within the system that disables notifications and presenting information to the user in alternative friendly ways.

Users tend to favour systems that carries out tasks on their behalf automatically, such as using spreadsheet software which is an alternative way of carrying out manual calculations, similarly this system will browse and retrieve information from the internet and make decisions on behalf of the user.

The method of deriving the systems functionality, deliberately tackles each problem this is done to assure that the system fulfils its aim as it provides a direct solution to the problem.

Based on the above information it is deemed that the solution presented tackles the problem directly by the set of functionalities it offers and achieves its objective.

### **2.5.2 Technology feasibility**

The following section is concerned with assessing why the use of Multi Agent Systems will be feasible to implement such system.

There are various recommendations within the literature review, suggesting that the use of Multi Agent systems is a good approach towards tackling problems related to distributed systems, as the proposed solution will also tackles an issue related to distributed systems then this recommendation will be taken into considerations.

Efficient use can be made of the capabilities of Multi Agent systems which contributes towards achieving the objective of the system, a justification to the use of capabilities is mentioned below:

- **Coordination:** one of the project requirements was that the actions of one agent can be based on actions of another, for example one functionality was if a user buys an item the selling agent can give the user information of how much they can sell that item for, therefore the actions of agents are coordinated with each other hence why this capability would be used.
- **Autonomous:** one of the main functionalities of the system is to automatically perform tasks which users undertake when making online purchases, as agents are capable of carrying out tasks automatically without user instruction then this capability of agent system will definitely be used within the development of the system.
- **Learning:** a learning algorithm can be developed to assist the agent in finding information on a particular site, as different websites advertise products differently.
- **Decision making:** as the system is autonomous it should not be dependent on users command to complete a particular task rather the agent should be able to decide for itself, for example the buying agent should make the decision whether to refer the user to a particular site or not based on information it has available such a reputation rating and suitable product value.

If further capabilities are to be used and are not mentioned within this section then they will be included within the analysis and design stage.

Based on the fact that the capabilities of Multi Agent system can be made use of efficiently then they can be deemed as a suitable technology to use for the development of the proposed solution.

A Desktop or web application does not have the capabilities of Multi Agent systems, and mainly bases its functionality and behaviours on direct user interaction meaning that they are user driven.

### 2.5.3 Technical Feasibility

**Compatibility with the organisation current technology:** The organisation that will make the system available for public users, will do so by making the system available for download through their existing website, it is the organisations choice to either make the system available to users for free or upon purchase.

In order to make the system available to download for users it must be hosted on the organisations website in the form of an executable file, where users can access this file in order to locally install the system on their machines.

There will be no information that would be feedback into the organisations system as the proposed system is local to each user, there is simply no requirement for doing so.

Based on the information above, whether the system can co-exist with the organisation current technologies is not a concern.

**Compatibility with user's workstations:** The system will initially be targeted towards the Windows OS users which is the most commonly used operating system (StackOverflow, 2015) this is the case for desktop and laptop users, therefore targeting this OS will increase the possibility of the systems compatibility with the users' workstations.

The hardware requirements for the software are not yet known, but the system ideally should not have demanding hardware requirements, also to increase the number of possible users.

If the organisation did not have technologies that can co-exist with the proposed system or not able to recruit a team of technical experts then the implantation of the system will not be technically feasible.

#### **2.5.4 Schedule feasibility**

A Gantt chart has been created which includes a milestones for different stages throughout the development of the system, the time duration for each stage has been set out in consideration of any contingencies which may delay the delivery date of the project, for example it may take one day to assess the aims and objectives of the project but instead it will be given two days.

The completion of the project by the defined date (04/04/2016) is possible, and the Gantt chart has been created accordingly which is included within the appendix section of this report.

#### **2.5.5 Economic Feasibility**

There may a number of charges which may occur if additional functionalities to the system are to be added this will be the case for maintaining the system and producing updates and patches, prices may vary depending on the nature of the additional functionality or update.

There may be other costs which are associated to the running of the system and these costs will be the wages of the technical experts within the organisation that will be in charge of providing technical support to clients.

### **2.6 Risk Assessment**

Within the following section, any potential risks which may arise during the development of the system or throughout its lifetime are predicted and included below with a potential solution that would eliminate or minimise the impact of each risk:

**Being categorised as malware:** Currently there is an enormous amount of malware hosted by many web sites which have malicious goals, one of the most common malware is AdWare which often repeatedly displays advertisements in the form of pop ups which randomly appear across the screen this tends to irritate any users, if the proposed system has similar behaviour then there may be a risk that users may consider the system as malware.

In order to mitigate such risk, the user must be ideally aware of the systems objective this can be avoided by introducing a brief tutorial when the system is installed on the users machine, also information outputted from the system to the user must not be presented in the form of popups or additional windows as doing so will drive the user to consider the system as AdWare and results in the system having a bad reputation and may be classified as AdWare by anti-malware software such as anti-viruses.

**Encountering fraud related sites** As the number of online shopping sites and purchases increase on yearly bases so does the number of fraud incidents relating to e-commerce (FFA, 2015). As one of the functionalities of the multi agent system is to recommend web sites which may be in the users interest, then there may be a risk that the system may refer the user to a fraud site which may result in the system being associated with fraud although this is not the intention.

Such risk can be mitigated if the agents only retrieve information from reputable site, the sites can either be predefined within the system or an algorithm can be implemented which will assist the agent in assessing whether the site can be trusted or not, ideally based on reviews consumer make on different review sites.

**Health and safety**, the system is software based rather than mechanical or physical therefore the only health and safety risk may be if the user is exceeding the amount of time they are spending in front of a computer screen.

**Security risks:** The system is not intended to hold any sensitive user data such as names, date of birth and bank details therefore there will be no risk of any users data being exposed to unauthorised access such as hackers or not being able to comply with the data protection.

## 2.7 Development tools and languages

It has been decided earlier that Multi Agent Systems will be the chosen technology to be used within the development of the system, however there are still further decisions to be made such as choosing suitable development tools and programming technologies, this section will include an analysis on the tools, technologies and methodologies that can be used throughout implementation.

It has been indicated and mentioned by various authors within the literature review that some of the development software is not as friendly as it should be, and the methodologies do not focus on all stages of the development, hence why this analysis is made to assure that the most suitable choices are made.

### 2.7.1 Development tools and software

Most of the development tools available are specific to a particular programming language e.g. JAVA, or/and a particular agent-oriented language such as AgentSpeak, therefore when it

comes to choosing a suitable development tool, the choice is not made independent of the programming language as they both are related and required throughout development.

The characteristics of development environments will be investigated, these characteristics are described below and their importance is justified:

- **Targeted platforms:** as stated earlier the proposed system is targeted towards windows platforms, therefore it's important to choose a tool that allows for the development of software that is compatible with the targeted platform.
- **User support:** it is always helpful to seek guidance from tutorials or manuals when support is needed rather than having to rely on trial and error.
- **Programming language:** when a development environment is chosen the choice cannot be made independent of the programming languages to be used, some development environments support the use of more than one language.
- **License:** although that some licenses are open source, but they may require the publishing of projects developed using that open source development tool onto online sites which may be against the clients' policies also it means that project work undertaken using that development kit will be available freely to others which isn't always desirable.
- **Methodology support:** some development tools limit the development of the system to a particular methodology such as AgentTool3, this is not desirable if the chosen methodology is different to the one provided by the environment.

There are other characteristics but they will not be discussed (such as maturity), as the above chosen characteristics seem sufficient to make a suitable decision

Although the development of Multi Agent Systems is not standardised but some tools and languages comply with the BDI model and FIPA standard, which are both discussed below as they continuously referred to throughout this stage and are of significance importance:

The BDI model has a great significance within the field of MAS, its brief description is as follows: The belief (awareness of the environment) desire (the goals of an agent) intention (it's plan towards accomplishing a particular goal), created to ease the development of agent systems, it also allows for agents to respond dynamically to environment changes, as each agent is aware of its role within the environment, its goals and how to achieve these goals through a number of plans. (Padgham, n.d.)

FIPA a standard that can be adhered to when developing agents that communicate with other applications externally rather than just creating an agent system where agents can communicate with each other only.

The following are the investigated development software's:

1. **Jason:** An interpreter of an extended version of the BDI based agent-orientated language AgentSpeak, where improvements are added, such improvements are the handling plan failures, often used to develop reactive agents (Developers, 2015)

2. **MadKit:** An agent development platform, which is built upon the OCMAS approach (Ferber, Gutknecht and Fabien, 2004), eliminating the need for a predefined model within the development tool making it more heterogeneous in regards any communication protocols and communication between agents (fmichel, 2016).
3. **AgentTool:** is a graphical based development environment available as a plugin to the Eclipse IDE, it implements the O-MaSE methodology from analysis to implementation, where the analysis and design stage is carried out by producing a number of models, these models describe the systems behaviour, it also identifies protocols, interactions, roles, goals and plans (Garcia-Ojeda, DeLoach and Robby, 2007).
4. **JADE:** the JADE development framework has been frequently used by authors referenced within the literature review, but it has not been stated why, possibly it could be that this is one of the friendliest tools available according to this survey (Kalliopi and Nick, 2015).

JADE's goal is to simplify the implementation of agent systems, the GUI tools provided within the environment play a crucial role in doing so, some of these tools are: the sniffer agent allows the developer to view the messages exchanged between agents, another tool allows to remotely manage agents either to stop or restart them, it is obvious that these tools will play a great role in debugging and troubleshooting the system (Kalliopi and Nick, 2015).

5. **JACK:** is a development environment that allows the development of a number of autonomous agents within one environment, it allows agents to be defined by their goals, knowledge and social belief (based on the BDI model), as each agent has its own goal it may communicate with other agents within the same environment in order to achieve that goal (JACK).

JACK uses two programming languages to develop agents, JAVA and JACK (an agent language), which is an agent-oriented programming language, JACK is an extension of the JAVA language, however a distinction between JAVA and JACK can be made by their syntax, any code written within JACK is compiled into JAVA. (Pierre-Michel and Demazeau1, n.d.) Provides details of the characteristics of the development tools introduced above.

The characteristics of the tools above that comparisons will be based upon are summarised into table 2.2:

**Table 2.2 Development tools comparisons**

<b>Tool Name</b>	<b>Targeted Platform</b>	<b>User Support</b>	<b>Supported Languages</b>	<b>License</b>	<b>Methodology Support</b>
<b>JASON</b>	Cross platform	Manual, tutorial, book, demos, mail support.	java and agent speak	open GL	Not limited to a methodology

<b>MadKit</b>	Cross platform	Demos, examples, online tutorials, online forum.	Jess, BeanShell, JAVA	GNU GPL	Not limited to a methodology
<b>AgentTool</b>	Cross platform	developers manual, tutorials publications	JAVA	Academic	O-MaSE
<b>JADE</b>	Cross platform	tutorials, examples and books	JAVA	Academic public	Not limited to a methodology
<b>JACK</b>	Cross platform	chat, demo, documentation, guides and tutorials	JACK (extends JAVA)	LGPLv2	Not limited to a methodology

### 2.7.2 Programming Technologies

Two types of programming languages are often used in the development of Multi Agent systems, these are agent-orientated languages and object-orientated languages.

Agent-oriented languages are often concerned with identifying certain characteristics and behaviours of Agent systems, such as goals, plans and interactions.

Although agent-orientated languages assist in identifying certain characteristics that define an Agent system, but they do not define how these characteristics are implemented, hence why object-orientated languages are used to implement these characteristics.

It is not a mandatory requirement to implement an Agent System using those two types of languages, as some environments do not require the developer to use two languages, such as AgentTool which only requires the use of JAVA.

There tends to be a close relation between Agent organisations and object-oriented languages, the close relation is due to the similarity between agent types and objects, as different agents can belong to different families, similarly different object types belong to different classes (Dastani, Jorge and Gomez-Sanz, 2005)

some practitioners (Dastani, Jorge and Gomez-Sanz, 2005) propose that agent systems can be implemented directly in an object orientated language eliminating the need for an Agent language, this is possible but it is simpler to defined agent behaviour at a higher level of abstraction prior to implementation, Agent-orientated languages also make it possible for implementation upon certain models such as BDI also to implement some sort of rationality into agent behaviour.

The languages introduced below are those which are supported by the development environments introduced previously:

- 3 **JACK:** An extension of the JAVA language to allow the development of agent systems, and the modelling of agent behaviours (Pty., 1999) it extends the JAVA language by the following: a number of added classes and methods as well as new syntax to distinguish between JAVA syntax and JACK syntax, eventually any code written in JACK gets compiled into JAVA.

The language consists of a number of constructs, some of it includes the following:

- **Agent:** this includes the behaviours of an agent, message exchange, the environment changes to respond to, communication to accept and plans it knows of to execute.
- **Belief set:** used by agents to gather information about their environment, it can be queried using logical members (used within logical programming languages).
- **Events:** events which may take place within the environment that agents can respond to.
- **Plan:** in order to achieve a goal a plan is to be followed, a plan may be a set of instructions to be executed in a particular order.
- These constructs are added to the JAVA language in the form of classes.

The language also introduces a number of statements: declarative used to declare queries and capabilities, reasoning statements, such as: achieve a condition or wait for an event.

- 4 **AgentSpeak:** An agent orientated-language based on the BDI model, used to describe the behaviour of an agent, combines the features of object-based languages and the features of the BDI model (Rao, 2000s), as the language is based on the BDI model it's constructs are very closely related to the model itself and these include (Duma, 2014):

- **Beliefs:** information available about the agent itself and agents within the environment and the environment it is a part of.
- **Goals:** the aims, roles or duties of the agent and their state, these goals can either be inner goals or public goals that can be invoked by other agents.
- **Events:** events that occur when an agent changes its goals or beliefs.
- **Plans:** how the agent is going to achieve its goal, a plan may consist of a number of goal statements.

Each Agent family created within AgentSpeak relate to a class created within an object orientated language, agents created represent instants (objects) of that class, agents developed within this language can be referred to as rational agents.

In addition the syntax can be easily understood and memorable, most of the syntax used is a single character only such as (?, ! And +).

Messages to be sent from an agent can either have a single agent receptions or an entire family of agents, in addition messages exchanged can be assigned with a priority level, and these messages can be exchanged over a WAN messages (Bădică, Budimac and Ivanović, 2011)



**5 JASON:** is an extended version of AgentSpeak, it inherits the features of AgentSpeak therefore it is also based on the BDI model and has the same constructs, JASON is supported by JAVA (Michael, 2012)

Jason does not modify the agent speak language rather it adds features to it, these features include:

1. The ability to respond to plan failures, an agent can have multiple plans that assist in achieving the same goal, these plans may not always be successful therefore the agent may need to attempt to execute another plan which may be successful.
2. There are a number of internal predefined actions for agents, these actions can be easily modified within JAVA to expand or modify their functionality.
3. Agents may be able to generate notes about their beliefs, these notes can be used by functions to be executed.
4. JASON is interpreted by the JASON interpreter, which can be in the form of a plugin to an IDE e.g. eclipse.

Table 4 shows the advantageous and disadvantageous of each language:

**Table 2.3 Comparison of development languages**

Language Name	Advantageous	Disadvantageous
JACK	The agent-orientated language is embedded within the object orientated language therefore does not require the use of two separate tools	Complex and requires a deep level of understanding before it can be applied
	JAVA and JACK syntax can be easily recognised	
AgentSpeak	Easily understood and memorable characters	There are a number of implementation of the language such as JASON
	The language relates closely to the BDI model, if the model is understood correctly then the language can be understood easily	
	Allows for the development of responsive agents	It is unclear how the tool translates diagrams into generated code and how efficient it will be
JASON	Allows for alternative plans to be executed in the even a particular plan has failed	Requires an understanding of agent speak before it
	Easily memorable and understood syntax	

### 2.7.3 Methodologies

Methodologies for developing software systems are often concerned with the entire development cycle from analysis to validation and verification, this does not tend to be the case for methodologies used for the development of MAS, some methodologies are concerned with the analysis and design stages only such as GAIA, while others are concerned with analysis, design and implementation stages such as O-MaSE.

A number of methodologies have been developed to assist in the development of MASs to make use of its features, capabilities and behaviours, software development methodologies can be quite general allowing them to be used for the development of a range of software systems, however they aren't suitable for the development of agent systems as they are not focused on the constructs of agent systems (Goals, beliefs), this has also been one of the reasons which driven towards the development of the GAIA methodology (Micheal and David, n.d.)

**The GAIA Methodology:** Is an agent-orientated methodology that focuses on the analysis and design stages of developing MASs, the developers of the methodology (David Kinny, Michael Wooldridge) describes the domain characteristics in which they believe their methodology can be applicable or suitable, these include:

- When the relationship between agents remains the same throughout runtime and does not change.
- The methodology is completely independent of a particular platform and language.
- When a system contains a number of agents less than a 100.
- The abilities of agents remain unchanged during runtime, as in not increasing decreasing or modified.

**The analysis stage:** as progression is being made through the analysis stage the implementation bias becomes greater, eventually reducing the number of implementation methods, it starts with an abstract level where the agent system is conceptualised but its inner running's are not clear yet, this divides the methodologies into two concepts abstract and concrete with an aim towards developing an understanding of the systems structure.

The analysis stage aims towards identifying certain constructs of the system:

- **Roles:** each agent within an organisation can be assigned with a role, for example in the system proposed earlier included two main roles those associated with buying and others associated with selling.
- **Permissions:** defines when the agent may be able to perform an activity, or who it can interact with
- **Protocols:** a set of rules of how can an agent perform such an activity how does one role interact with another
- **Responsibilities:** can be similar to the agent goals
- **Activities:** similar to plans

**The design stage:** After an abstract model is derived from the analysis stage, then the design stage can assist in decreasing the level of abstraction in order to easily implement the system, with an aim to clarify how agent interact with each other in order to achieve their goals.

The GAIA methodology assist in decreasing the level of abstraction to a level where the use of object-orientated methodologies is possible, meaning that GAIA is not sufficient to complete the design stage but it makes it possible for object-orientated methodologies to do so.

Upon completion of the design stage three models should be available:

- **Agent types:** where agents that have similar roles can be grouped and categorised into one type, the aim of doing so is to reduce the number of instantiated agents during runtime as it beliefs that the resources each agent consumes is high therefore by reducing the number of agents it optimises the agent system.
- **Services:** correspond to a set of plans an agent may use to achieve its goal, these services can be in correspondence to the activities defined in the analysis stage, a number of service can correspond to one activity, the inputs and outputs to these activities can be defined in the protocol within the analysis stage.
- **Acquaintance:** used to define the communication pathways between agents, and whether they the communication is a two way communication or not, as in if the agent sends and receives information, or just sends information

### **The O-MaSE methodology:**

An object orientated methodology which aims towards allowing designers to construct a custom methodology that meets their requirements, it provides developers with a repository of method fragments, the developer has the option to chooses the suitable fragments and construct them together according to the methodologies guidelines, upon doing so a new instance of the O-MaSE methodology would have been created.

This is known as situational method engineering where engineers are given the flexibility to innovate and develop a set of engineering methodologies that suit their requirements.

The O-MaSE methodology can be used for the analysis, design and implementation stage of developing a MAS.

The methodology can be implemented within the development environment AgentTool, through the use of available graphical tools.

Different version of agent tool implement different versions of the O-MaSE methodology, currently the latest is agent tool 3 which contains improvements made to the older versions of the methodology.

Table 5 includes the fragments (activities) available within the repository, each fragment has a number of activities associated with it, a developer has the flexibility to either use some of these fragments or all of them, each task has a deliverable.

**Table 2.4 Fragments of the O-MaSE methodology**

<i>Activities</i>	<i>Tasks</i>	<i>Work products created/modified</i>	<i>Responsible method-roles</i>
Requirements gathering	Requirements specification	Requirements spec	Requirements engineer
Problem analysis	Model goals	Goal model	Goal modeller
	Refine goals		
	Model domain	Domain model	Domain modeller
Solution analysis	Model organisation interfaces	Organisation model	Organisation modeller
	Model roles	Role model	Role modeller
	Define roles	Role description document	
	Define role goals	Role goal model	
Architecture design	Model agent classes	Agent class model	Agent class modeller
	Model protocols	Protocol model	Protocol modeller
	Model policies	Policy model	Policy modeller
Low level design	Model plans	Agent plan model	Plan modeller
	Model capabilities	Capabilities model	Capabilities modeller
	Model actions	Action model	Action modeller
Code generation	Generate code	Source code	Programmer

#### 2.7.4 Choices Made

Based on the information presented above AgentTool (O-MaSE) will be the chosen methodology and development software, the justification of making such choice is included below:

- Greatly eases the implementation of the system, as it produces a number of models throughout the design and analysis stage in which can be easily implemented, most of these models are graphical and require little knowledge to be understood.
- Identifies all aspects relating to the system: the models generated as a result of implementing the methodology can allow for most related aspects to the system to be identified such as the interactions between agents and the protocols they comply with as a rule of interaction.
- It eliminates the need for an agent orientated language as the methodology is available to be graphically implemented through a plugin that automatically generates code from the models produced by the developer.
- Quickens the development process: as it eliminates the need for an agent-orientated language it will fasten up the development process as the requirement to learn a new language no longer exists.

Methodologies such as the GAIA methodology uses different constructs to those which are used by an agent-orientated languages, this may difficult the implementation stage as the developer would have to relate the constructs which are used by the methodology to the language, as O-MaSE eliminates the need for an agent-orientated language such potential issue will also be eliminated.

Although it does not support other stages within development but it does greatly ease the development processes as discussed previously, other methodology can be used through later stages such as the testing stage.

Reliability: the system is available as a plugin to a mature IDE (Eclipse), the IDE is continuously supported by updates and a wide range of support option is available such as forums and user support.

## 2.8 Analysis (O-MaSE)

The following starts implementing the analysis stage defined by the O-MaSE methodology which mainly consists of the following two stages:

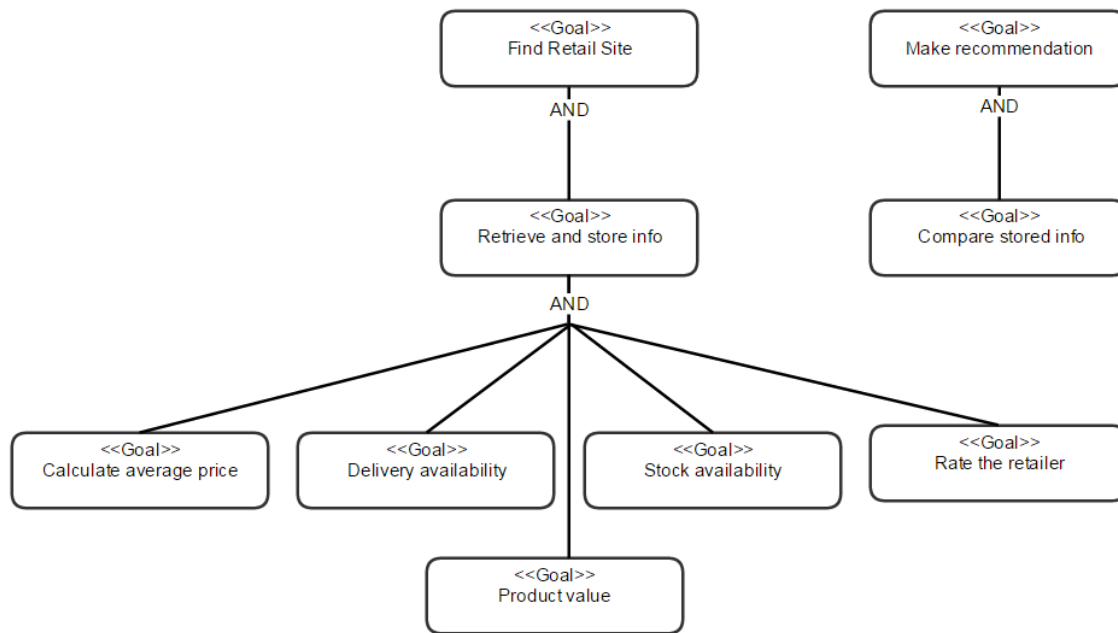
### 2.8.1 Problem analysis

The problem analysis stage within the O-MaSE methodology is concerned with identifying information about the environment and products of the environment, it consists of a number of tasks:

**Model goals (task):** the purpose of this task is to convert the requirements of the system into goals, identified goals can be of two types (master-goals and sub-goals) the relation between these two type of goals can be OR or AND refined, OR refined means that the master goal can be achieved if one or more of the sub goals are achieved, however AND refined goals require all sub goals to be achieved in order to achieve the master goal,

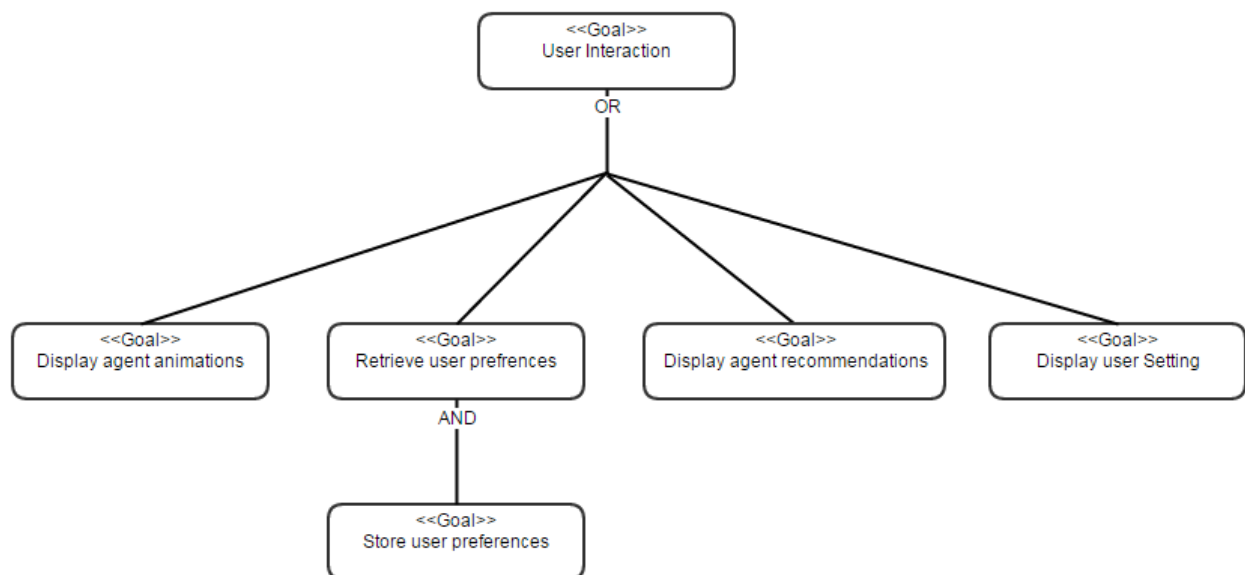
Upon completion of this task a goal model would be produced as shown in figures 2.1 – 2.5, the lines between goals represent the relation and the text indicates how the goals are refined.

Figure 2.1 represent the goals relating to the Buying agent, these goals are gathered from the requirements document, it shows that the agent must identify a site where an item is sold then retrieve information from that site and store it in order for it to perform calculations later that will allow the agent to recommend a product.



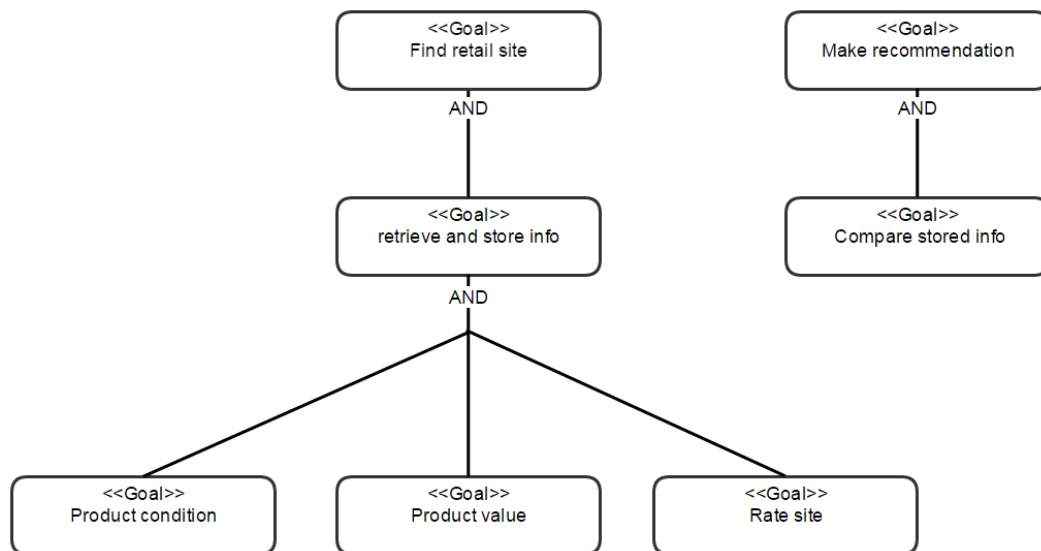
**Figure 2.1 The buying agents' goal model**

Figure 2.2 shows the goals of the graphical user interface (GUI), it shows the goals that the user interface fulfils, the main goal of the GUI is to enable interaction between the system and the user, an example of its sub goals may be to display and retrieve information.



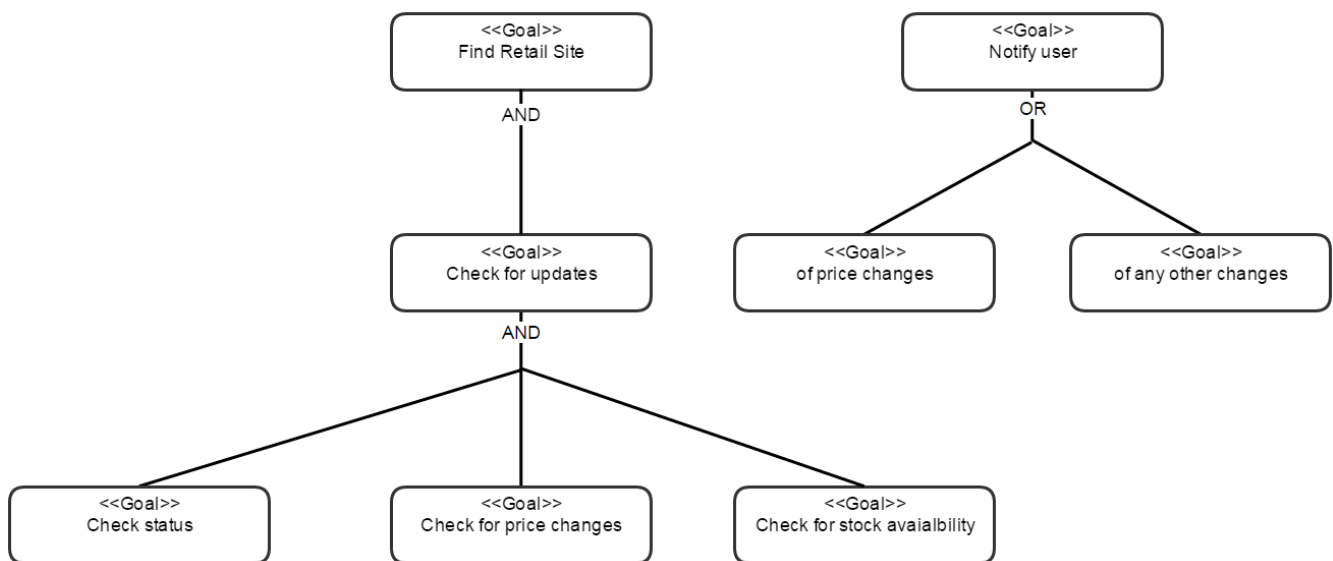
**Figure 2.2 The GUI's goal model**

Figure 2.3 represents the goals of the selling agent, just like the buying agent it identifies a site and retrieves the information it requires from that site and stores it for later computing and processing.



**Figure 2.3 The selling agents' goal model**

It is mentioned previously that a wish list would be included as part of the system, the wish list checks on daily basis for changes within the status of items defined by the user, if any changes are detected then it will notify the user of these choices, figure 2.4 shows the goals associated with the wish list:



**Figure 2.4 The Wish list Goal Models**

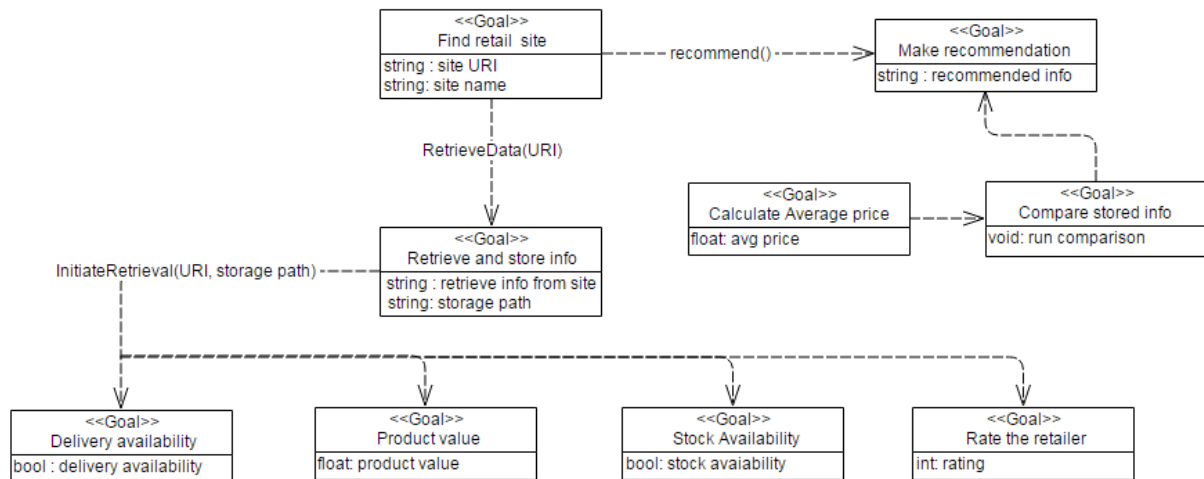
**Refine goals (task):** can be used to capture the dynamic aspects of the system by identifying the events that trigger goals and to identify the parameters associated each goal, as well as to identify the order of goal execution as some goals are preceded by others.

The O-MaSE methodology creates an instance of a goal whenever that goal is required as there can be scenario where not all goals are required.

Figures 2.5 – 2.8 show the parameters associated with each goal, for example the product value goal in figure 2.5 has a float attribute associated with it, when goals are preceded by others

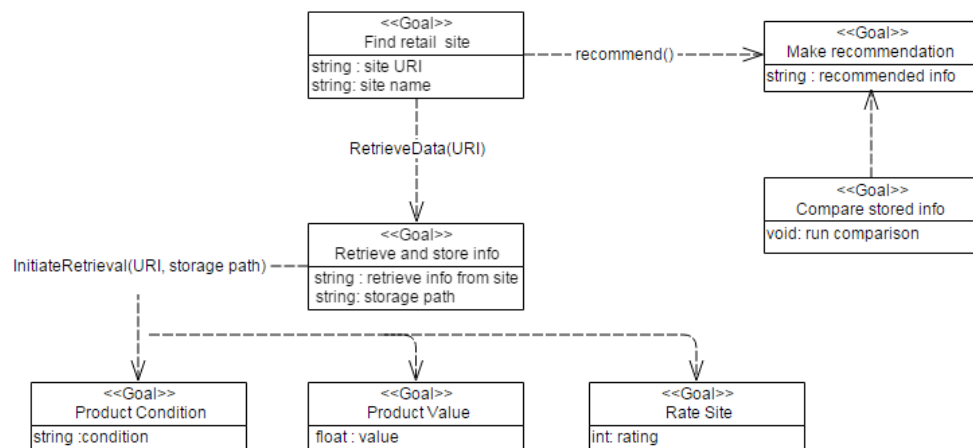
there will be an arrow to indicate so, if text is included on the arrow it indicates the events which trigger a goal.

Figure 2.5 refines the goal model associated with the buying agent, it has two main goals when the Find retail site goal is achieved then the next main goal would be to make a recommendation.



**Figure 2.5 The buying agents refined goal model**

Figure 2.6 refines the goal model associated with the selling agent, it works on the same principle as the buying agent refined goal model, where events trigger goals and goals can be preceded by others.



**Figure 2.6 The selling Agents refined goal model**

Figure 2.7 refines the user interface goal model, the main goal of the GUI is to achieve user interactions, which can be achieved by either displaying information passed from agents or when the user sets or modifies preferences.



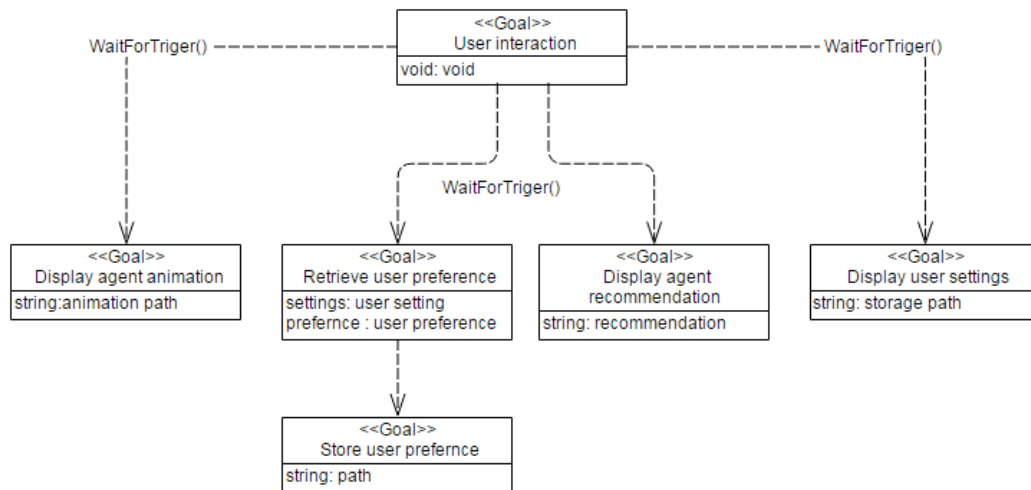


Figure 2.7 The GUIs refined goal model

Figure 2.8 refines the goals associated with the wish list its main goal is to check for changes made to the status of products within the wish list then notify the user of any changes, as indicated from the figure the wish list sub goal compare the old values with new ones to determine if a change has occurred.

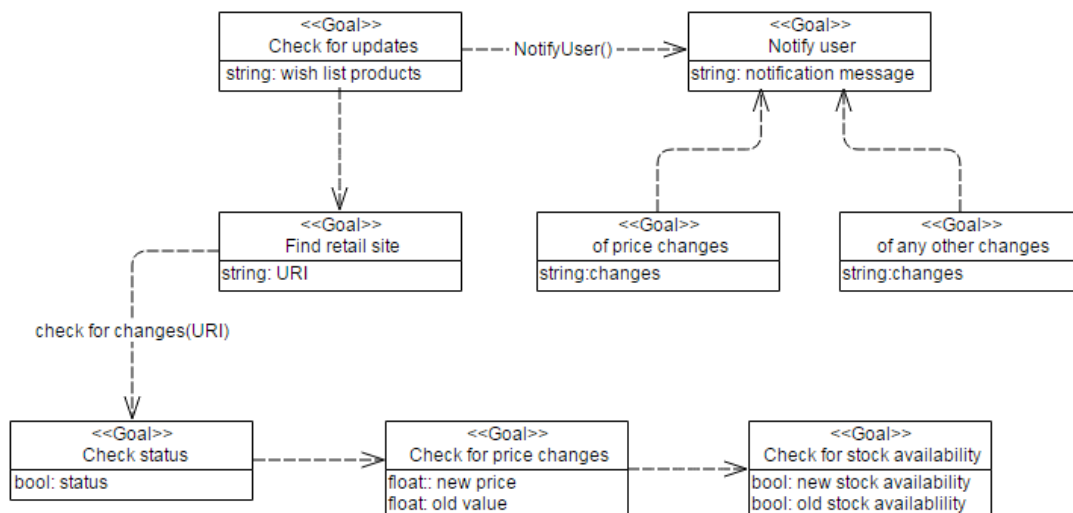


Figure 2.8 The wish lists refined goal model

## 2.8.2 Solution analysis

The purpose of the solution analysis fragment is to identify interactions between the system and external entities, this is achieved by modelling the organisations interfaces and the roles within the organisation

**Organisation interface model:** The methodology states if the system developed is a sub system to another, then the interactions between the subsystem and the main systems are to be modelled otherwise if the system is a standalone system then the external interaction (such as system/user interactions) are modelled, as well as interactions with other system entities (such

as databases), modelling these interactions will allow for protocols to be developed within the design stage to define the rules of interaction.

Figure 2.9 shows the system interaction model, the system consists of different components and entities which all interact together in order to achieve its aim, the whole system revolves around agents. Agents retrieve the user preferences once they are added to the storage then use those preferences to retrieve information from the internet and store them in order to process them later.

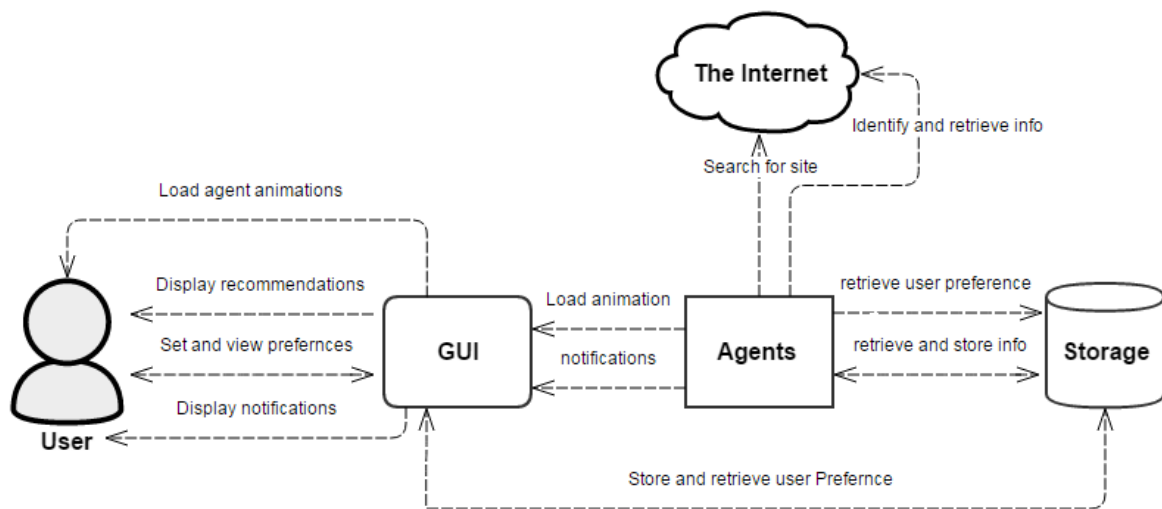
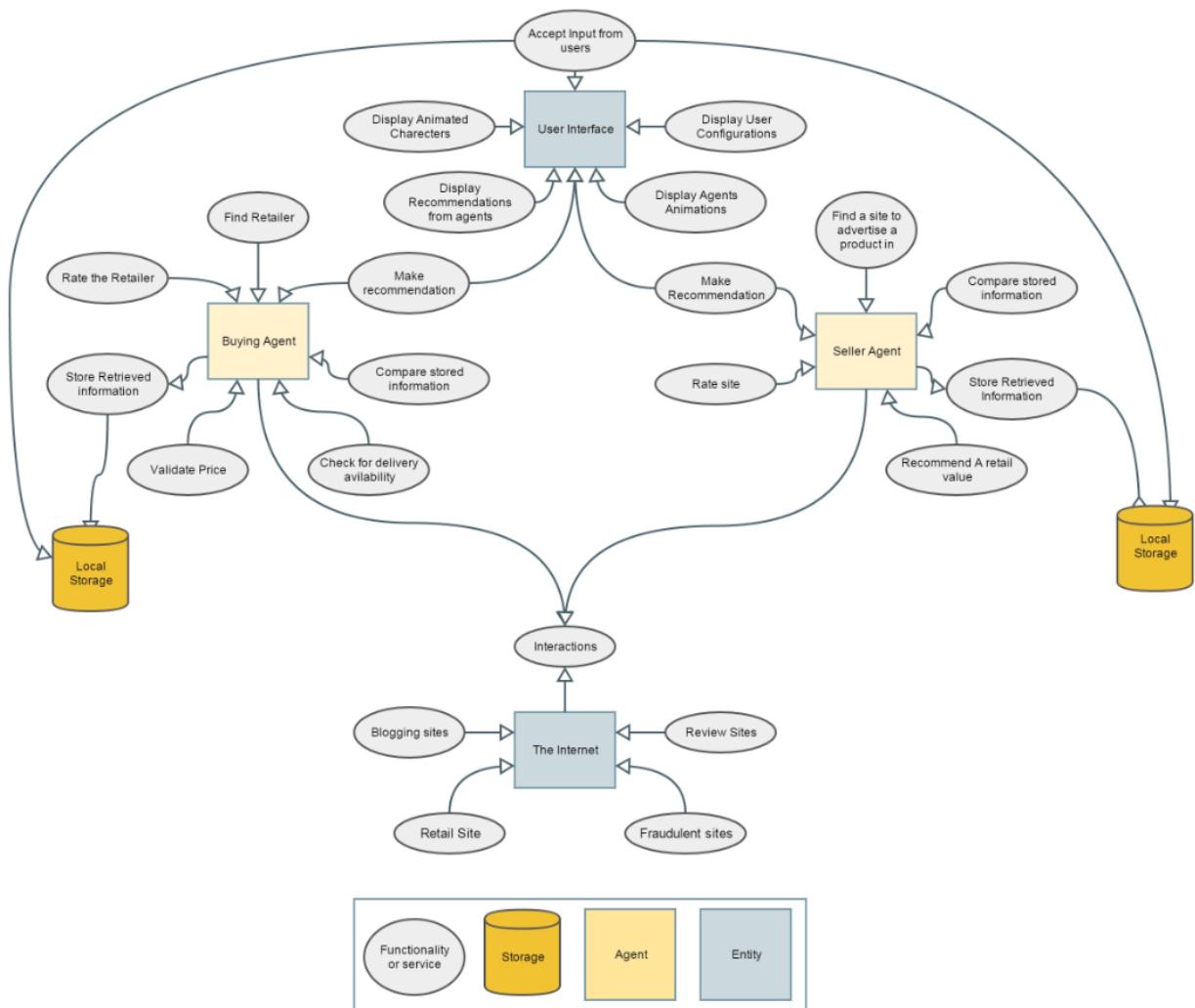


Figure 2.9 the organizations interface model

## 2.7 Preliminary design

Figure 2.7.1 shows the number of components which may take part within the system or relate to it, these components are included within squares, the circles that are linked to squares represent the functionality of each components or what is hosted by it.

The design will provide information of the functionalities of the system as well as what may link these components together.



**Figure 2.7.1 Preliminary design**

# Chapter 3 Design

This stage consists of two sub design stages, one design stage is concerned with the design stages defined by the O-MaSE methodology, while the second stage is concerned with the design of system entities that are not covered by the O-MaSE methodology such as the Database design.

## 3.1 Database design

It has been mentioned previously that a storage method is required, therefore a SQL Database will be used to store information which agents continuously deal with, as shown in figure 3.1.

The contents of the Database are derived from the information required to be stored within the system and the details derived as a result of generating the various models during the analysis stage, the purpose of each table will be justified below:

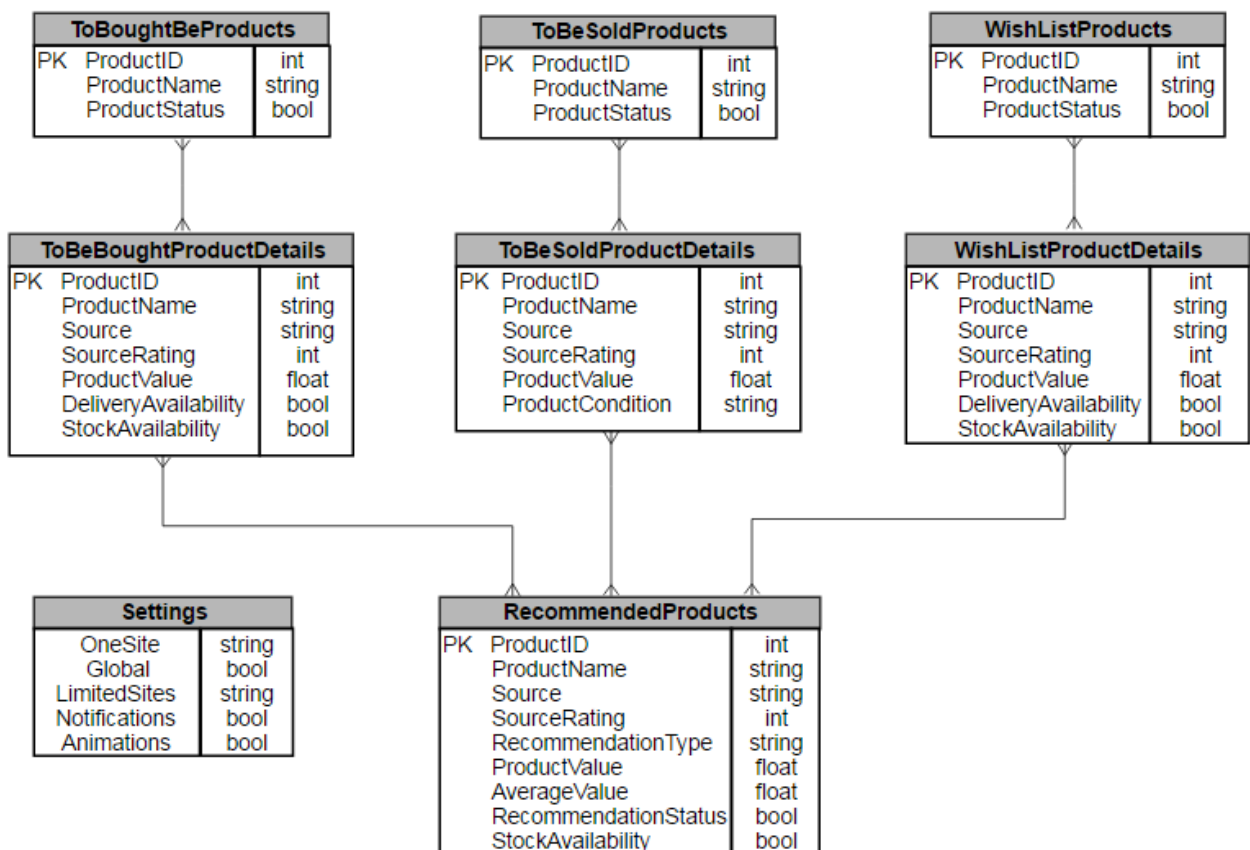


Figure 3.1 Database contents and table relations

Figure 3.1 shows the data and types of each entry that will be stored within the databases tables. The following description below justifies the purpose of each table.

The *ToBeBoughtProducts*, *ToBeSoldProducts* and *WishListProducts* tables in figure 3.1 all achieve the same goal, which is to store the name of the product the defined by the user, each agent will refer to one of these tables to identify which products is the user interested in.

The *WhishListProductDetails*, *ToBeSoldProductDetails*, *WhishListProductDetails*, tables in figure 3.1, also achieve the same goal, which is to store the information retrieved from different site such as product value, and the source of where this information was retrieved from.

The *RecommendedProducts* table in figure 3.1, is used to store any recommendations made by the agent and its attributes.

The *Setting* table in figure 3.1 is used to store the users' preferences such as disable animations and notifications, or to limit the agents' interactions to one or a few sites rather than dealing with all sites globally.

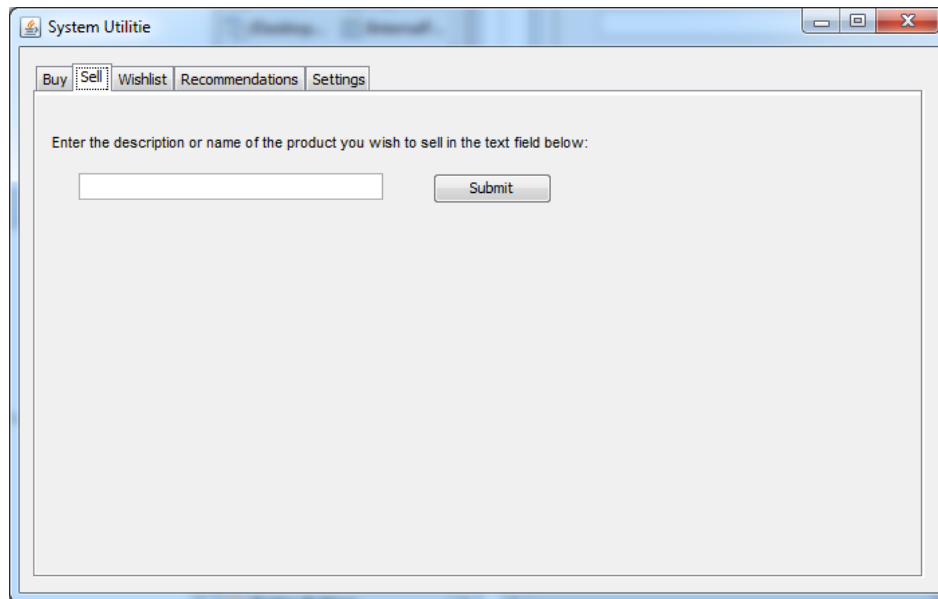
## 3.2 Graphical User Interface (GUI)

As mentioned in the requirements document a GUI is required to achieve several purposes:

The GUI mainly consists of:

1. Windows forms: to allow for user /system interactions
2. Animated character and behaviours: these will be included within the design just yet as they can be added in the finished product and not the prototype.

The Windows form to be used will have the shape of the form shown in figure 3.2 where mainly one form is used with separate tabs, each tab has should fulfil the purposes stated below:



**Figure 3.2 shows the structure and layout of the GUI to be implemented**

Figure 3.2 shows the design for the sell tab only, however details of how each tab is implemented will be provided in the implementation section as the GUI design tool used is the same as the implementation tool.

The *Settings* tab: the main purpose of the setting tab is to allow the user to set their desired preferences, the number of preferences has been reduced to what is necessary to reduce complexity as user often find software's that are full of preferences to be unfriendly.

The *Recommendations* tab: once the agent is ready to recommend info to the user, it will display the recommended info within the recommendation tab, the use of a single tab to display recommendations within is done deliberately, as mentioned previously in the risk assessment that recommendations should not be displayed in the form of popups as this is commonly used by ad-ware and other malicious software, therefor by displaying recommendations in a dedicated area, the user will not be distracted and irritated by information presented them, this fulfils the requirement of presenting info in a user-friendly way.

The *buy* tab: the main purpose of this tab is just to allow the user to input details about the product they are willing to purchase, the only detail that is required from the user is the products name e.g. Paper white Kindle 16GB.

The *Sell* tab: the main purpose of the sell tab is to allow the user to input details about the product they wish to sell, these details will be stored within the database.

The *Wish List* tab: the wish list tab will give the user the option to add products to the wish list as well as to remove existing products within the wish list, it also displays all the current wish list items within a table.

Each and every single tab within the form has a help option associated with it, briefly summarising the purpose of the tab and how the use may interact with it.

### 3.3 Architecture design (O-MaSE)

The architecture design is one of the stages of the O-MaSE methodology, this stage focuses on capturing the details of the systems high level components. Details captured at a high level are later used to implement lower level designs, for example the implementation of agents' communications (a lower level component) have to be in accordance with the communication protocol (a higher level component).

This stage includes a number of activities, only one activity out of the three activities will be used within this project as the others are simply not required:

#### 3.3.1 Model protocols

The purpose of the model protocol activity is to produce a protocol that governs the rules of interactions between agents, therefore the following protocol has been created:

##### Communication protocol

The design will later state how communication between the two agents will implemented, when it will occur and how agents respond to messages exchanged, in order to implement communications efficiently between agents, then a communication protocol is required to govern the rules of interaction, therefore the following rules below have been created, each with a specific justified purpose:

It is stated in the design that XML files will be used to implement communication between agents, therefore you may find that some of the rules found below relate directly to how the XML files are created and how they are handled.

- All XML files generated by agents should have the same XML version (1.0) and encoding standard (UTF-8) for the purpose of maintaining consistency.
- All generated communication files will be stored within a dedicated directory, as the dedicated directory would be the path which the agent use to check for any incoming messages and places outgoing messages.
- Communication files generated by agents must have one of the two possible names, the file name should either be *ToBuyingAgent* OR *ToSellingAgent* as the file name indicates which agent should process the message.
- There should be no spaces used within the file name, XML tags and tag contents this is also done to preserve consistency.
- The number of files contained within the dedicated directory shall not exceed two files, this is done to reduce space and increase performance as the agents will have less files to process, this can be achieved by deletion of files once they have been processed.
- Tag contents: it is important that each communication file contains the tags stated below, as these tags will be used by the agents to interpret the files contents:
  - The *<CommType>* tag: used to hold the communication type e.g. message exchange or negotiation, this will notify the agent of how to interpret the contents of the file.

- The *<CommStatus>* tag: used to indicate whether a message has been read or not, this status is change from not-read to read once the intended receptionist has read the message.
- The *<CommValue>* tag: this contains information that is dependent on the communication form taking place.

Communication between agents occurs in two scenarios, message exchange and negotiation, both scenarios will comply with the rules of the protocol stated above.

### 3.4 Low Level design (O-MaSE)

The low level design stage is another stage defined within the O-MaSE methodology. This stage is concerned with the production of a low level design which details how each goal/functionality/requirement is achieved, therefore details within this stage can be used throughout the implementation stage.

The stage consists of a number of activities, however only two out of the three activities will be used as the third is not required. The two activities include the plan model and the actions model.

#### 3.4.1 Plan model

This activity is concerned with transferring the goals of the system into plans, plans included within this activity will be at a high level as another activity will be concerned with specifying how these plans are implemented at a lower level.

The plans included within this section are derived directly from the goal and refined models created during analysis and they are also derived from the requirements document therefore they achieve functional and non-functional requirements as well as the systems goals.

Each plan is highlighted in bold below, and will be reviewed during the validation and verification stage and compared against the requirements and goals:

**Agents' identification of products in users' interest:** One of the agents' main functionalities is to gather information about products that the user is interested in, but prior to doing so the agent must identify which products the users are interested in and what information they require the agents to gather for them.

In order for each agent to identify which products it should deal with and how, the *ToBeSoldProducts*, *ToBeBoughtProducts*, *WishListProducts* tables, within the database have been created, each agent interacts the relevant tables, for example the selling agent interacts with the *ToBeSoldProducts* table.

The GUI can be used by the user to input products into the system then the agents can deal with the product details provided, this is the interaction between the user and the system, as shown in figure 3.3.



As indicated from the database design, each entry made to the *ToBeSoldProducts*, *ToBeBoughtProducts*, *WishListProducts*, contains a status attribute, this status attribute is used by agents to indicate whether they should deal with a product or just ignore it. This attribute is changed normally by agents this is done to avoid assigning the user with additional tasks than what is not necessary.

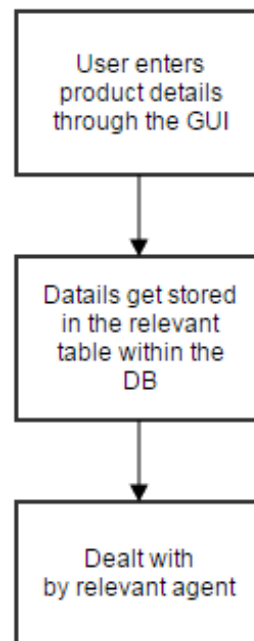


Figure 3.3 how agents identify a relevant product

**Data retrieval of products in the users' interest:** one of the main functionalities is to gather information about products which are in the users' interest.

As the previous plan allows agents to identify which product the user is interested in, then the agents can surf the internet and retrieve information about those products which explains why the *ToBeSoldProductsDetails*, *ToBeBoughtProductsDetails*, *WishListProductsDetails* tables within the database, their purpose is to store information gathered by agents about these products such as product price, retail site.

The agents gather their information from websites hosted on the internet, but prior to doing so they will check the user preferences as one of the user preferences is to limit the agents interaction to a single or multiple site rather than a global interaction, therefore if the user limits the agents interaction to ([www.ebay.com](http://www.ebay.com)) then agents can only base their interaction with that site otherwise the agents may continue gathering data from different sites on the internet until they could not identify any more sources.

Details of how this information is identified within sites and retrieved is included within the low level design.

Figure 3.4 shows the sequence of activities performed by an agent in order to achieve this plan.

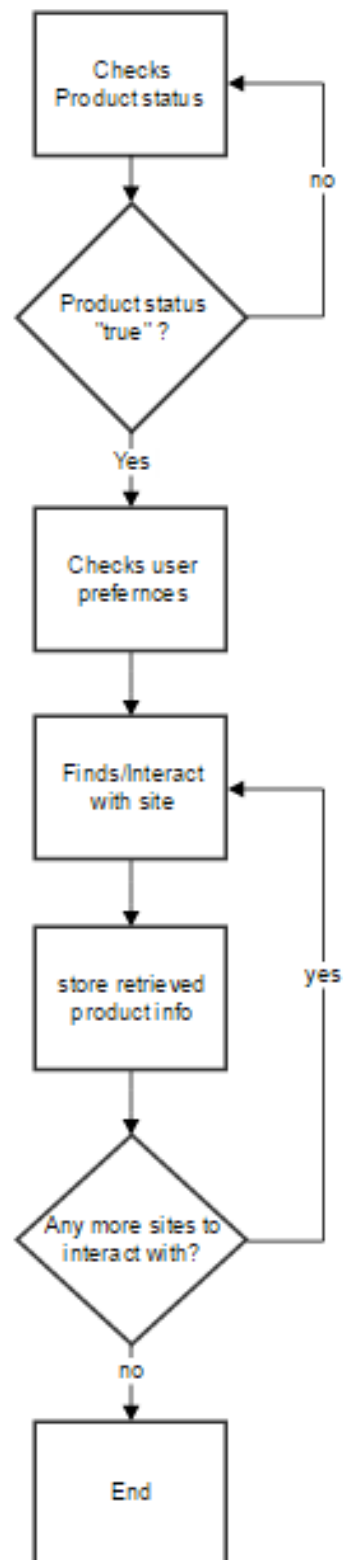


Figure 3.4 Plan for data retrieval and storage

**Calculating average price:** a common functionality between the two agents is to calculate the average price of products stored within the database. The two agents use the average price calculation in order to assist them in making the most suitable recommendation to the user.

Each entry within the *ProductDetails* tables of the database has a column which holds the product value of each product, this information will be used by agents to calculate the average price of a products, as agents will refer to all table entries that contains the same product name and refer to the product value column within the same row.

The functionality that allows for this plan to be implemented will be discussed within the low level design.

**Comparison of products within the database:** as mentioned in the requirements document that one of the agents functionalities is to make recommendations to the user, there are mainly three types of recommendations which the agents will make as shown below, each recommendation is based on a number of calculations and comparisons of data retrieved by agents:

1. **Recommendation by the buying agent:** each time the user wants to purchase an item they will inputs the items details OR name into the system, from there on the agents will browse the internet for details about that item and stores them which explains why the data retrieval plan is included previously, once the agent collects information about the product it will make a recommendation to the user, based on the comparison below informing them that the information presented to within the recommendation would be the best place where you can buy that item, the comparisons are on the following bases:
  - The product value is reasonably close to or below the average price.
  - The source must have the highest rating.
  - The product is available for delivery and in stock.
2. **Recommendation by the buying agent (Wish list):**as detailed previously the functionality of the wish list is to notify the user of any changes made to the products they wish to purchase, these changes can be price drops or stock availability status change, these changes are to be detected by the following plan(figure 3.7):
  - If the products delivery status have changed then the user should be notified.
  - If the products has dropped in value then the user also has to be notified.

On daily bases the agents must retrieve data about the products within the wish list from the same sites they have been initially retrieved from that's if the product already exists within the wish list otherwise if the product has been recently added to the wish list then the agent just simply search the internet for where the product is advertised and retrieves the relevant data and stores it within the data base, the agents should compare the data available on the site with the one stored within the data base, this way if a change is detected then the agent will notify the user.

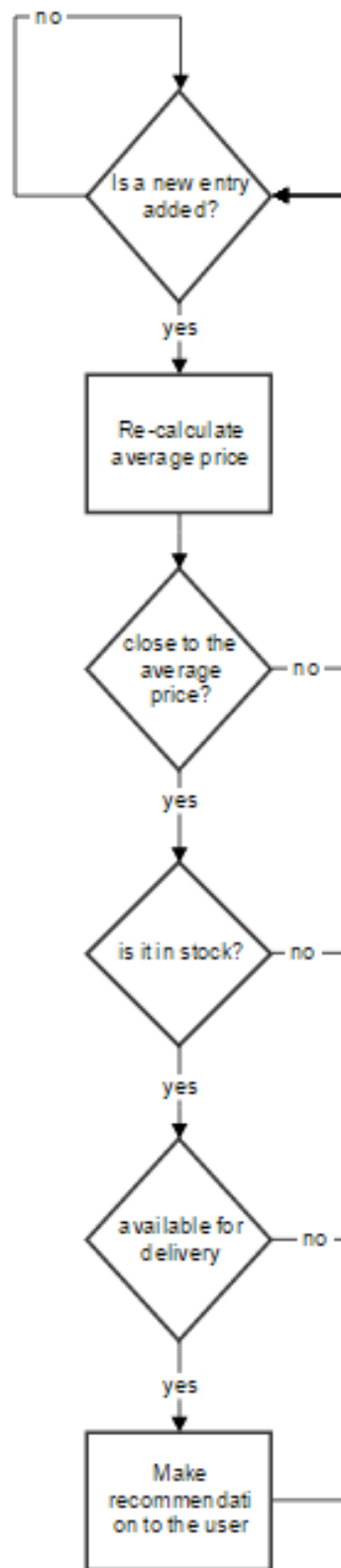


Figure 3.6 the buying agents recommendation plan

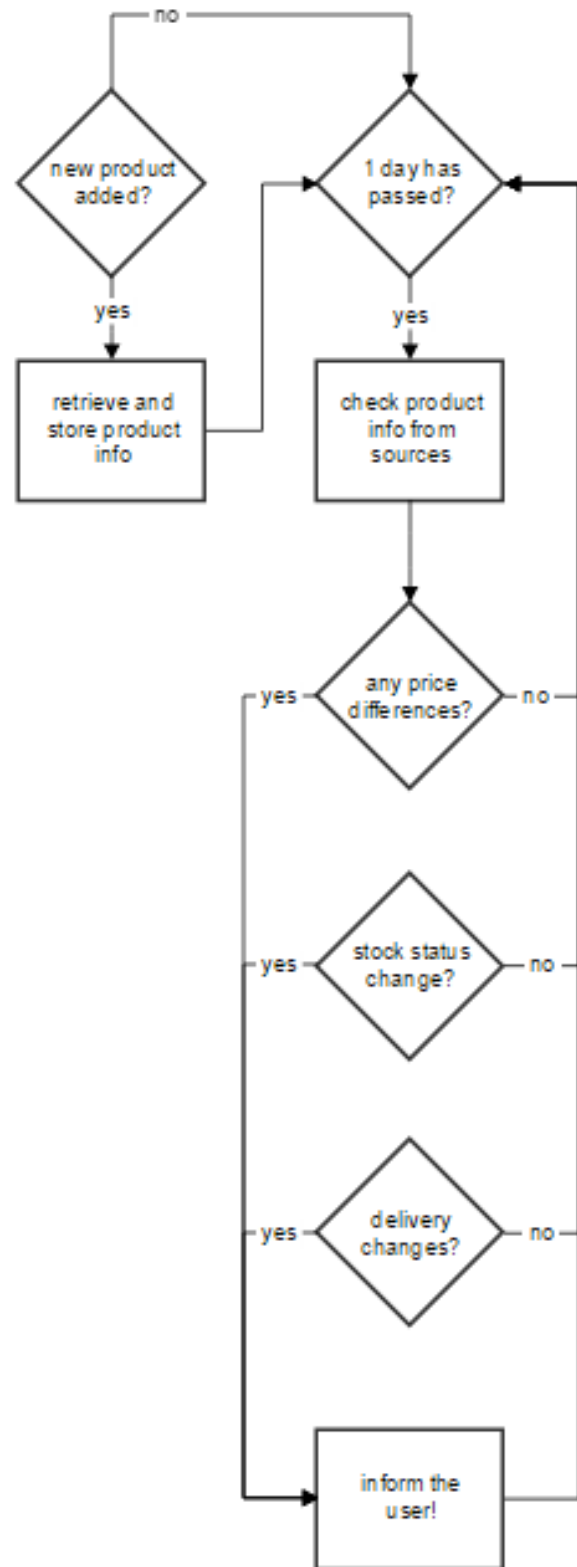


Figure 3.5 the wish lists recommendations plan

3. **Recommendations by the selling agent:** the recommendations made by the selling agent are concerned with providing the user the information required for them to sell a product, such as the best suitable price to sell the product at, and where to advertise it, this agent will also base its recommendations on information it retrieved from different sites, these tasks which the agents performs are shown in figure 3.7 and listed below:
- It calculates the average price of products and presents it to the user as the suitable sale price.
  - It compares the rating of sites and recommends the user to advertise their product at the highest rating site.
  - An additional feature which can be added is to provide two recommendation one that is based on the products condition being pristine and another on being used.

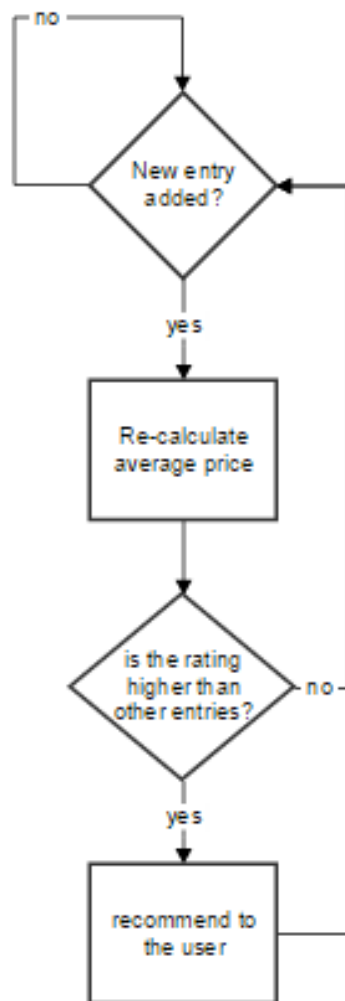


Figure 3.7 Recommendations plan for the selling agent

Each agent will perform these comparisons on daily basis and each time a new entry is added to the *ProductDetails* tables.

The bases of which the agents perform their comparison upon has now been defined but it has not been defined how this comparison is performed within the system but it is included within the low level design of the system.

**Notification production:** it is one of the requirements that the system produces notifications on the following occasions and of course when the notification user preference is not disabled:

- When an agent produces a new recommendation.
- When a change is detected within the wish list.

Notifications exist for the purpose of grasping the users' attention to a specific event. How these notifications are produced is detailed in the low level design.

**How a site is rated:** this functionality is required to assure that the sites to be recommended to user are not fraudulent sites.

The method of which a site is rated is simple, there are a number of sites online which provide site ratings based on users reviews and other aspects, the agents may interact with those already existing site and extract the rating of the relevant site.

**Data retrieval from sites:** a number of sites use existing technologies to allow interactions with other systems, these include web services, APIs and HTTP requests, in which queries can be sent to the targeted site and responses contain the data requested, therefore these technologies can be used as one of the methods to interact with sites.

There is also an alternative, the mark up script for any site can be easily viewed by anyone, often by examining the mark-up script language for the sites layout the developer would be able to identify how information is made available on the site therefore they may write a tool to extract the data.

There are no plans to include this functionality within the prototype but it will definitely be included once the system has been completely developed.

**Communication plan:** Communication between agents is one of the aspects of multi agent systems that is crucial to any multi agent system consisting of interacting agents, communication may be in many forms, some have already been discussed within the literature review, and they include negotiation and argumentation.

Communication between agents can be achieved in a number of ways, for agents distributed among networks communication may take place by HTTP requests while agents within a local system communicate by XML messages or through a database, as the agent current system consists of local agents then XML will be used to implement communications.

Within this proposed systems two forms of communication are mainly used, negotiation and message exchange.

There are a number of scenarios where communication will take place between agents, according to the requirements document:

1. When the buying agent recommends a product to the user: it is mentioned in the requirements documents that each time the buying agent makes a recommendation it will communicate the product details to the selling agent, therefor the selling agent will

inform the user that if the user decides to sell this product in the future they can potentially use the information provided to them by the selling agent, this communication will take place between the two agents in the form of messages, these messages will be in the form of XML files, once the selling agent receives a message it will simply added it to the *ToBeSoldProducts* table and deals with it as a product that has been added by the user.

2. When play fight begins between the two agents: it is mentioned previously that agents will occasionally play fight in the purpose of entertaining a user. Once the agents begin to play fight with each other they may eventually come to a stop, in order to implement this the negotiation aspects of agents will be implemented where agents will negotiate the appropriate time for them to stop play fighting, there will be a plan detailing the rules of negotiation. Negotiation will also be implemented in XML file (message) exchange

Using XML provides the freedom of creating custom communication protocols and custom message contents therefore it will be used to implement message exchange between files, in a later low level design stage, the structure of the messages will be as shown below:

File name = ToAgentType (e.g. ToBuyingAgent, ToSellerAgent)

```
<?xml version="1.0" encoding="UTF-8"?>

<CommType>      < CommType/>      // e.g. negotiation/message exchange
<CommStatus>    < CommStatus/>    // message status e.g. read/unread
<CommValue>     < CommValue/>     //e.g. product info/ negotiation
value
```

Messages are to be sent and received by simply placing them within the dedicated directory (as stated within the communication protocol), once the agent receives a message and acts upon it will change the status column to read, once this status has changed the sending agent deletes the message (file) from the directory this can also be considered as an acknowledgment mechanism.

The *CommType* tag will be used to inform the receiving agent how to deal with the message, by stating the type of communication or the purpose of it.

### 3.4.2 Actions model

This activity is concerned with defining how plans can be achieved at a lower level, and how they can be implemented within the programming language of choice.

Below are a number of low level plans detailing how the plans designed at a higher level can be implemented, some plans that have been included previously in the design may not be found as one of the low level designs below, this will be due to one of the two reasons, that the design is not to be included in the prototype or the design is too simple to be explained at a lower level:

**Interactions with XML files:** it is mentioned that communications will be implemented through the use of XML, which means that there must be a method in which interactions can be made between the system and the XML files.

There are mainly two methods of interacting with XML documents, the SAX model and the DOM standard:

1. The DOM standard: is a language independent standard used to establish communication with XML documents, the DOM standard loads the contents of the XML document into memory in the form of a tree structure where all the contents of the XML file is considered as a nodes whether it was comments child tags or root tags, this allows the developer to access the contents of tags independently and enable the programme to act upon it, it also allows for writing and modifying the XML.
2. The Sax model: the SAX model also allows for coms to be established with XML documents however it does not load the contents into memory, it simply reads the file and notifies the programme of what it has read, it reads the file in a sequential manner start to end, meaning that once one tag is read the developer may not read it again which isn't suitable for dynamically changing files.

The DOM standard is preferred over the SAX model and will be used to access XML files due to the following reasons:

- Although the contents of the file get uploaded into memory in the DOM standard, it may be thought that it would occupy large amounts of memory and cause issues, however this will not be the case in this MAS systems as the contents of the XML are just a few tags.
- The SAX method only allows for data to be read and not written to the XML file which goes against the requirements of the systems according to the coms plan as it requires reading and writing to the XML files.
- The SAX method is much more complicated than the MOD standard and requires a deep understanding before user which gives the MOD standard another advantage.

Figure 3.8 is an example of the DOM standard implementation, it demonstrates the simplicity of how data is retrieved from a file by making reference to the tag names



```

try {
    DOMParser parser = new DOMParser();
    parser.parse("mydocument.xml");
    Document doc = parser.getDocument();

    // Get the document's root XML node
    NodeList root = doc.getChildNodes();

    // Navigate down the hierarchy to get to the CEO node
    Node comp = getNode("Company", root);
    Node exec = getNode("Executive", comp.getChildNodes() );
    String execType = getNodeAttr("type", exec);

    // Load the executive's data from the XML
    NodeList nodes = exec.getChildNodes();
    String lastName = getNodeValue("LastName", nodes);
    String firstName = getNodeValue("FirstName", nodes);
    String street = getNodeValue("street", nodes);
    String city = getNodeValue("city", nodes);
    String state = getNodeValue("state", nodes);
    String zip = getNodeValue("zip", nodes);

    System.out.println("Executive Information:");
    System.out.println("Type: " + execType);
    System.out.println(lastName + ", " + firstName);
    System.out.println(street);
    System.out.println(city + ", " + state + " " + zip);
}
catch ( Exception e ) {
    e.printStackTrace();
}

```

Figure 3.8 Example code for the use of the DOM API in JAVA (Bruno, 2011)

**Interactions with DB:** as indicated from the plans included in the previous design stage that in order to achieve most goals, database interactions are required where data is continuously retrieved and processed.

SQL databases and java programmes are considered as two separate systems, just like most programming languages, often the developers of the language provide a number of libraries or API's to allow database/language interactions.

For example C based languages such as C++ and C# can use LINQ to SQL, however JAVA offers its own API (JDBC), just like any other data base API, the connection and login parameters to the data base must be defined somewhere and then a method of executing SQL queries.

The beauty of the JDBC API is that it allows the developer to hardcode raw SQL queries which makes the API simple and easy to use unlike the LINQ to SQL method where specific statements have to be learnt, another aspect which makes the API more appealing, is that after the SQL query is executed, the data is returned to the programme and can be dealt with as a primitive data type.

The sample in figure 3.9 demonstrates the simplicity of the API, its similarity with other existing DB APIs and its implementation:

```

public void DoConnect( ) {
    try {
        //CONNECT TO THE DATABASE
        String host = "jdbc:derby://localhost:1527/Employees";
        String uName = "Your_Username_Here";
        String uPass= " Your_Password_Here ";
        con = DriverManager.getConnection( host, uName, uPass );

        //EXECUTE SOME SQL AND LOAD THE RECORDS INTO THE RESULTSET
        stmt = con.createStatement();
        String SQL = "SELECT * FROM Workers";
        rs = stmt.executeQuery( SQL );

        //MOVE THE CURSOR TO THE FIRST RECORD AND GET THE DATA
        rs.next();
        int id_col = rs.getInt("ID");
        String id = Integer.toString( id_col );
        String first_name = rs.getString("First_Name");
        String last_name = rs.getString("Last_Name");
        String job = rs.getString("Job_Title");

        //DISPLAY THE FIRST RECORD IN THE TEXT FIELDS
        textID.setText(id);
        textFirstName.setText(first_name);
        textLastName.setText(last_name);
        textJobTitle.setText(job);
    }
    catch ( SQLException err ) {
        JOptionPane.showMessageDialog(this, err.getMessage());
    }
}

```

Figure 3.9 Example code of the use of the JDBC API (homeandlearn, 2015)

**Notifications production:** notification are required when certain events take place within the system that the user must be notified of.

It is stated on the Microsoft MSDN site that the one of the purpose of the task bar is to allow for notifications generation without irritating the user, according to this by using the task bar to display notifications a number of functional and non-functional requirements would be achieved, as one of the requirements is integrating a method of generating notification while presenting them in a non-irritating or distracting way to the user.

Therefore a task bar icon will be used to deliver notification these notifications would just be a few words indicating the purpose of the notification such as “An item within your wish list has decreased in price”.

The MSDN Online resources provide an SDK detailing how notification can be generated as well as clear guidelines as to how to use these notifications and implement them within a system.

Before the system generates any notifications it should check whether notification are disabled or enabled within the settings window.

**Compliance with user preferences:** As indicated from the plans within the previous design stage that the outcome of most plans is dependent on the user preferences, for example the system would not produce any notifications if the notifications options have been disabled.

The most efficient way to allow for functions to check the value of the current setting is to create a class that is dedicated to providing this information, therefore a class will be created that always has the users settings loaded into it from the data base, the class will offer a number of functions that can be called within other classes if they required knowledge of the current setting.

**Date tracking:** some functionalities of the system are dependent upon time and date changes, in order to implement this within the programme the primitive java.util class can be used which provides details about the current date and time.

The date tracking functionality will be used mainly to change the status of products within the ToBeBoughtProducts, ToBeSoldProducts tables, when the product have existed in them for a single day, products within those tables that have a true status means that the agent will keep having to collect information about them from the internet, it goes against the aim of the system to allow users to change these status as it will just add another responsibility/task for them that they may not understand therefore it will be done by agents.

**How comparison is implemented:** there are mainly three types of comparisons performed prior to an agent making a recommendation, it is worth noting that each recommendation made by agents is stored with a table within the database, the database will make contain one recommendation entry for each product and no more than one, this is done deliberately as the agent should make one recommendation to the user not to confuse the user by bombarding them with option, the comparison types are listed below:

1. Buying agent comparisons: as indicated from figure 3.5 that the comparison of products prior to recommendation can be achieved by carrying out a series of checks that if all has a positive output then a recommendation is made, these series of checks can be implemented with embedded if-statements and DB interactions using the JDBS method, the comparison functionality will be triggered each time an entry is added to the database
2. Buying agent comparisons (wish list): as indicated from figure 3.7 each time a day passes, the agents will refer to the sites it has already retrieved information from and it will compare the information available on the site to the previous-information available within the database, if a change is detected than the user is notified of these changes.
3. Selling agent comparisons: easily indicated from figure 3.6 and requires no further explanatory explanation.

### 3.5 Class model

The functionalities and plans designed to achieve goals can be implemented within classes, this will allow the designer to take advantage of the object orientated aspect of the JAVA language in order to simplify and organise the implementation of the system.

As shown from the class diagram (figure 3.10) each class packages a number of closely related functionalities that can be used by other classes, it shows that the DB interactions class gets used by the agents' class and the UI class.

There are common functionalities between the two agents, each agent is represented as a class however they both inherit from the same class which contains the shared functionality such as rating a site, similarly there are two classes in charge of handling communication, the message exchange class and negotiation however they both share the need of having to deal with XML file hence why they inherit from a super class (parent).

As shown in the diagram, each class has a description associated within detailing the functionalities to be included within, and the relations between classes is also shown

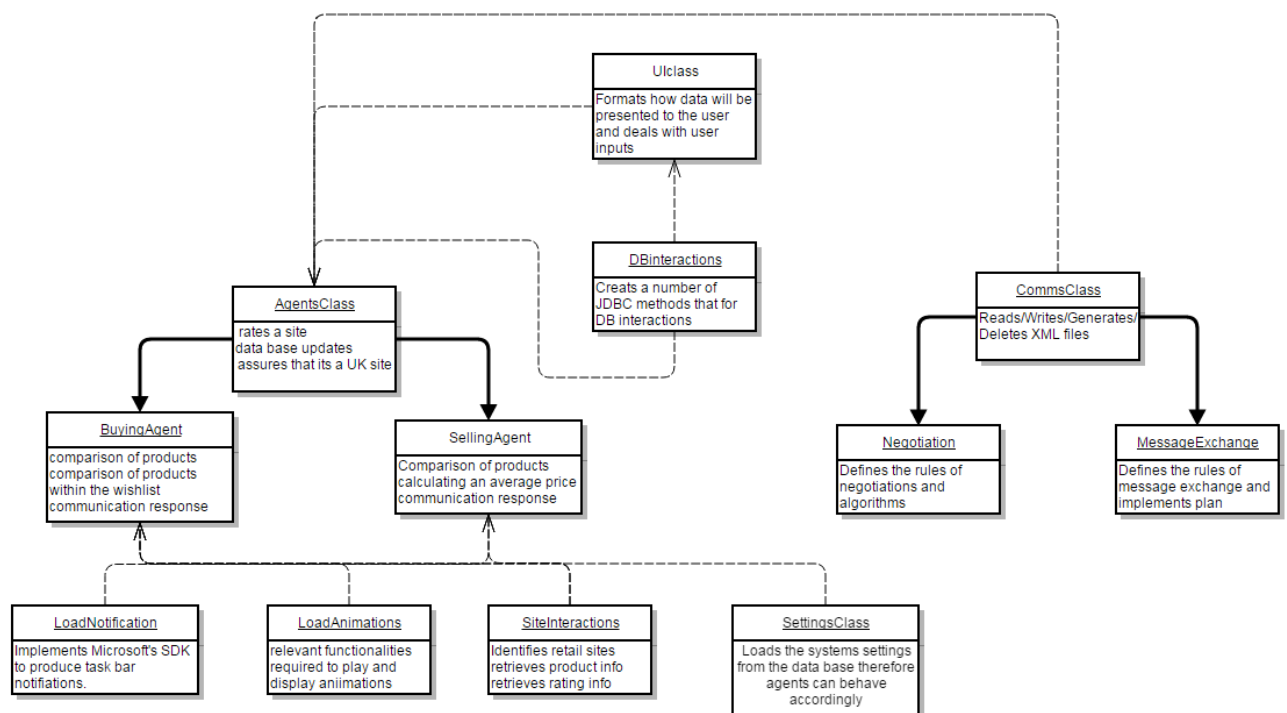


Figure 3.10 showsa the organisation of functionalities into classes

## 3.6 Agents Aspects

A number of plans and functionalities have been designed in order to implement the goals of the system, however it has not been discussed how these functionalities relate to the aspects of a MAS, therefore this section relates the functionalities to aspects.

These aspects are what defines an agent systems, without them the system may be another non-intelligent system that is completely dependent upon user interactions and instructions.

The aspects below have been examined within the feasibility study and chosen to be used within the system:

**Autonomy:** the agents within this system are autonomous and non-dependant on the user instructions to determine which functionality to perform, this is best demonstrated by the model in figure 3.11, it shows how the agents are aware of changes within the environment and what changes they respond to.

**Communication:** communication method, plans and protocols have been defined earlier within the design stage as well as the purpose of the communication and its relevance to the functionalities required within the requirements document.

**Coordination:** communication between agents is what drives coordination as there are some actions that will not be carried by an agent unless it receives some sort of indication from the other agent, an example of this would be how an agent will not play fight back with an agent until it started.

**Learning:** according to the current design method, the agent is instructed how to read and retrieve data from sites as well as identify the site type (e.g. retail), however this may not be very practical, as different sites display data differently meaning that the agent has to be instructed how interpret and read each site individually, which is a time consuming/never ending task as the number of web-sites is tremendous.

This can be solved by implementing a learning algorithm in which the agents can use to interpret sites independently by referring to the learning algorithm this will also allow for an increase in the quantity of sites in which the agent interacts with, however this functionality is a field of research and a topic in itself and would be time consuming to include within the proto-type therefore it will be left out for now and can be added any time in the future.

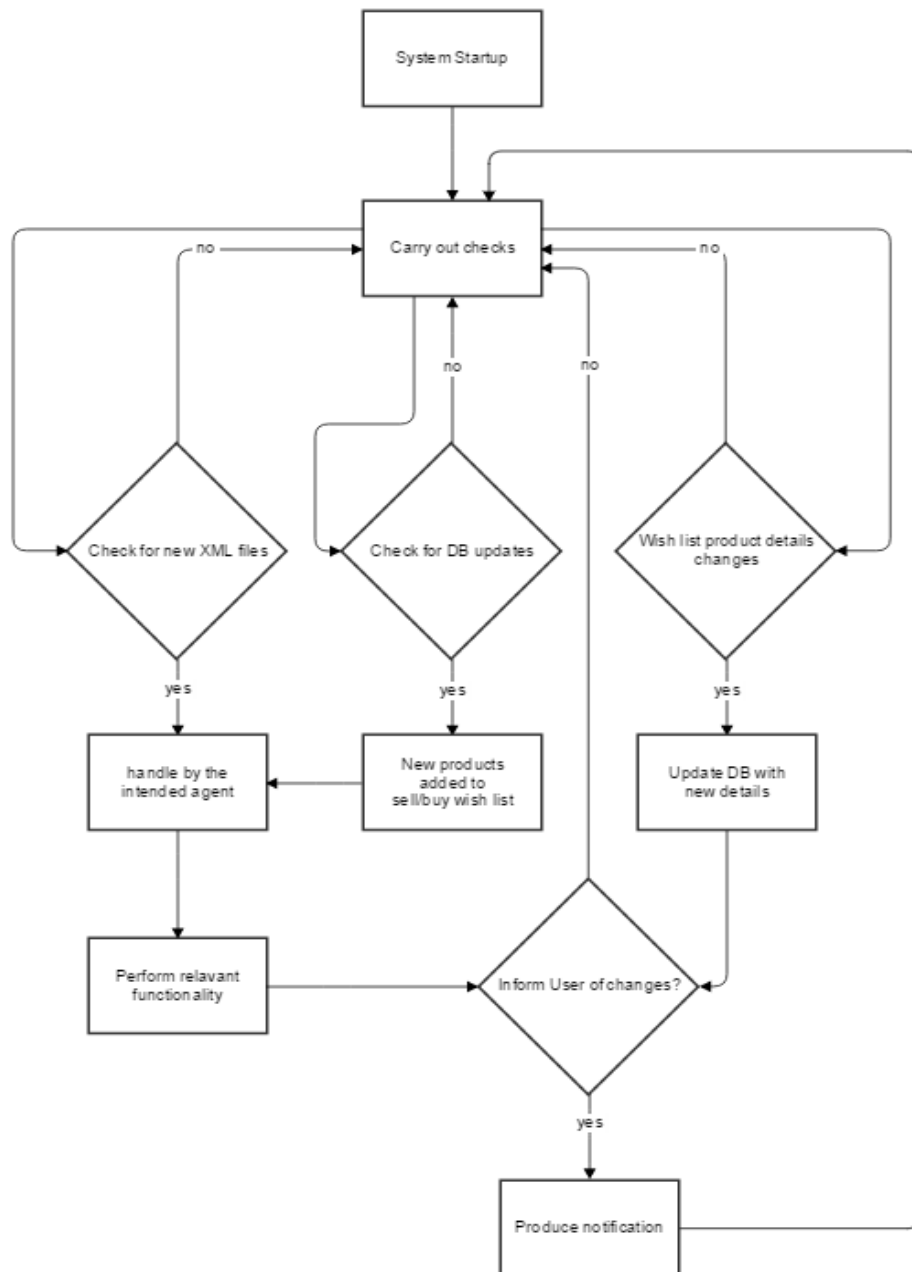


Figure 3.11 Demonstrates the autonomous behavior of agents

# Chapter 4 Implementation

The previous analysis and design stages of development provided thorough details, detailing how the goals of the system can be achieved through a set of plans defined at high and low levels of abstraction, these plans will now be used as a guideline to implement a prototype.

The implementation of the prototype is not according to the previously followed O-MaSE methodology as the methodology does not state any guidelines for implementation.

An open source IDE (Eclipse MARS.1) is the main development software used, however there are other software's required throughout the development, these will be introduced in their relevant sections of this report.

The prototype will be developed in a way that it potentially can become a complete product if more functionalities are added to it, as the already existing functionalities within the prototype do not have to be modified or re-created, therefore the more functionalities are added to it, then the more of a complete product it becomes.

After the implementation and testing is complete then the verification and validation stage will provide details of the functionalities that have been included and those which have been left out.

## 4.1 Database implementation

As mentioned within the design stage that a relational database will be implemented, the type of database to be used must be compatible with the JAVA language:

A number of a data bases have been investigated to assess their suitability and compatibility with JAVA and the ODBC API:

- SQL Server Database: has to be hosted on a server to be used, as the system is local to each workstation then there is no need for a server.
- My SQL Database: is compatible with the JDBC API however it has to be hosted on a server.
- Apache Derby Database: a database that can be embedded into a JAVA project and is compatible with the JDBC API however it does not provide a GUI to implement the DB rather it provides a console application which may be more human error prone and slows the development process.
- MS Access Database: can be stored locally and is compatible with the JDBC API, its DBMS provides a user friendly GUI which eases and quickens the development process of the database.

Therefore the database of choice is the MS Access database, the database has been implemented identically to the model available within the design as shown in figure 4.1, the

same case is for the relations between tables which exist to assure that consistency is maintained between all tables and is also shown in figure 4.1, the database columns have data types identical to the ones included in the design

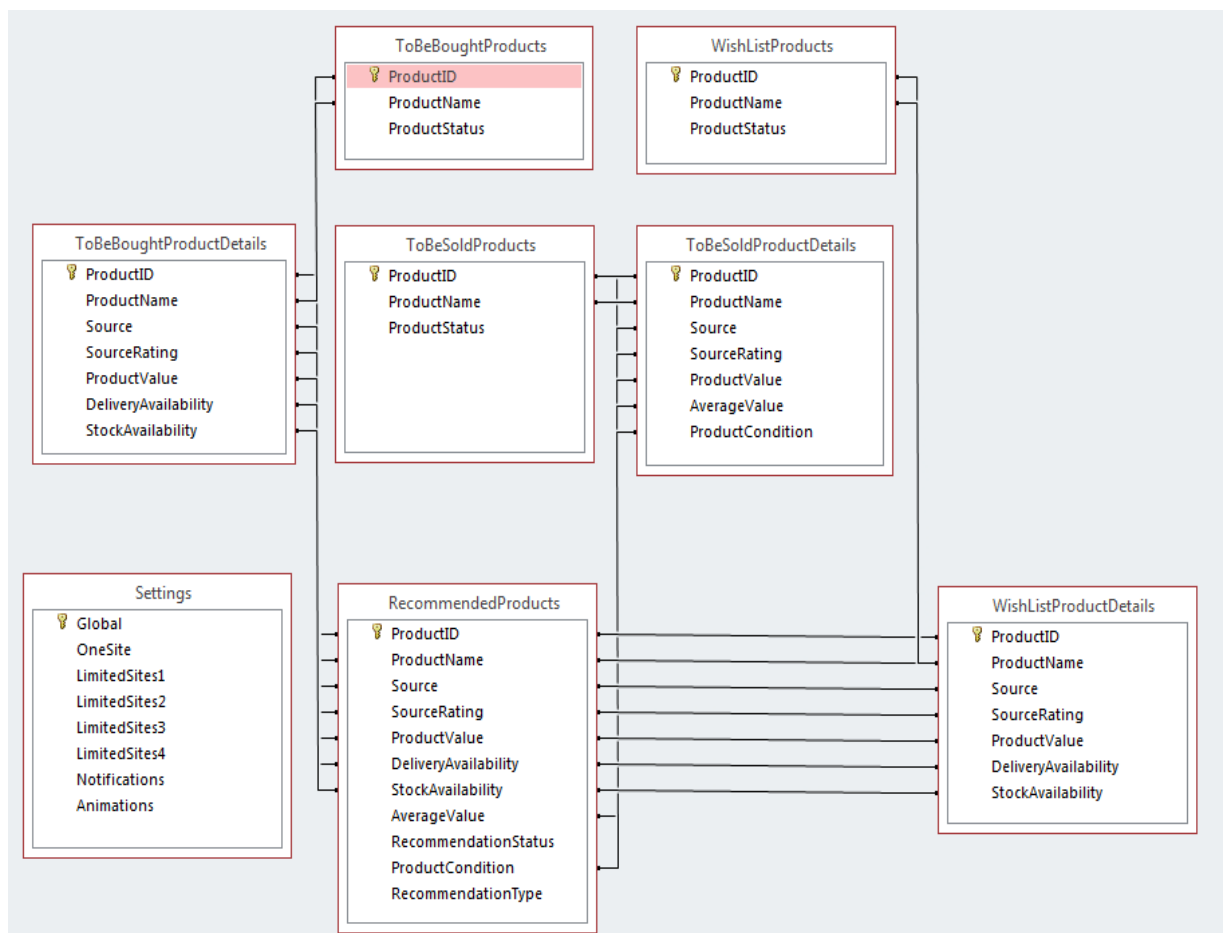


Figure 4.1 the DB implementation in MS Access

## 4.2 Class model implementation

This stage, implements the classes identified within the class model in the design section. Each class packages a number of functionalities within it, these functionalities are an implementations of the plans discussed at a high/low level within the design stage.

### 4.2.1 The database interactions class

As indicated throughout the analysis and design stage that different entities within the system are directly and indirectly continuously interacting with the DB, as the type of interaction is common between these entities then it is only reasonable to package the these interactions as functions within a class.

The class uses the JDBC API to interact with a database along with the *ucannaccess* library which allows interactions with an MS Access DB, the JDBC API provides a connection object



which can be used to establish a connection with a database through the use of a URL as shown below:

```
final private String ucannaccessString =  
    "jdbc:ucanaccess://h:/FYP/Implementation Files/Project Files/AgentsDataBase/AgentsDataBase.accdb";  
  
Connection conn =DriverManager.getConnection(ucannaccessString); // DB connection
```

In order to represent SQL statements within JAVA, the JDBC API provides another class called *Statement*, the object representing the SQL statement gets linked to the connection object, therefore once a statement is to be executed it can execute for the DB specified within the connection object:

```
Statement s = conn.createStatement(); // declaring a statement object (from JDBC lib)
```

The statement object provides many execution methods, each has a particular purpose as shown below:

- **ExecuteQuery:** Executes a statement that may return a single or multiple values.
- **Execute:** used to execute a statement without expecting anything to be returned, mainly used by insert statements.
- **ExecuteUpdate:** used to execute update or delete statements.

The *ResultSet* is also one of the classes provided by the JDBC API, its purpose is to store any values returned as a result of executing a SQL statement, depending on the SQL statement executed, multiple results may be returned, in such event a while loop can be used to read through the values

```
ResultSet rs = s.executeQuery(SQLst);  
  
while (rs.next()){  
    // do something  
}
```

The above are the main features of the JDBC API that will be used throughout this class.

The class *DBinteractions* have been created, it contains a number of functions, each function can be publically invoked, it only contains a single private variable which is the connection string, this variable gets used within all functions and has a static value:

```
final private String ucannaccessString =  
    "jdbc:ucanaccess://h:/FYP/Implementation Files/Project Files/AgentsDataBase/AgentsDataBase.accdb";
```

The class contains three functions, each of those functions fulfils a purpose as shown below:

1. A function that executes a SQL Statement that returns a value.
2. A function that executes a SQL without returning a value.

### 3. A function that executes a SQL Update Statement required for delete operations

Each one of these functions has a string parameter, this parameter is used to pass in the SQL statement to be executed, this is done intentionally to simplify the use of this class and speed up the development process therefore each time a DB interactions is to be made, the developer would simply use this class without worrying about creating connection and statement objects, all that is to be done is add a SQL statement of choice.

The following function executes a statement without returning a result:

```
// Executes a statement without returning a result
public void ExecuteDBQueryNoReturn(String SQLst){

    try{
        Connection conn =DriverManager.getConnection(ucannaccessString); // DB connection
        Statement s = conn.createStatement(); // declaring a statement object (from JDBC lib)
        s.executeUpdate(SQLst); // execute a statement without returning anything
        s.close();

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

The following function executes a SQL statement which returns a result, the result gets stored within a variable of type *ResultSet* and gets returned out of the function:

```
// executes a SQL statement and returns the result of the execution
public ResultSet ExecuteDBQuery(String SQLst){

    try{

        Connection conn =DriverManager.getConnection(ucannaccessString); // DB connection
        Statement s = conn.createStatement(); // declaring a statement object (from JDBC ;
        ResultSet rs = s.executeQuery(SQLst);

        return rs;

    }catch(Exception e){
        e.printStackTrace();
        return null;
    }

}
```

The only difference between the following function and the previous one within this class is that it performs an *executeUpdate* method which is used by update and delete statements, the function does not return any values:

```
// can be used to pass statements that contains Delete or Update Functionality
public void ExecuteDBQueryUpdate(String SQLst){

    try{
        Connection conn =DriverManager.getConnection(ucannaccessString); // DB connection
        Statement s = conn.createStatement(); // declaring a statement object (from JDBC lib)
        s.executeUpdate(SQLst);
        s.close();

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

#### 4.2.2 UI Class implementation (GUI)

Another main component of the system is the GUI used to implement Agents-User interactions. In order to implement windows forms within the Eclipse IDE a plugin was used (WindowBuilder) which provides GUI development tools similar to the ones provided by visual studio and generates code for UI componenets automatically once they are added to the design.

An example of automatic code generation: once UI componenets are added using the graphical tools provided by window builder, the code below gets generated automatically, it declares basic information such as the name, type and coordinates of the componenets:

```
final JRadioButton ToBeSoldRB = new JRadioButton("To Be Sold");
ToBeSoldRB.setBounds(277, 77, 109, 23);
RecommendationsTab.add(ToBeSoldRB);
```

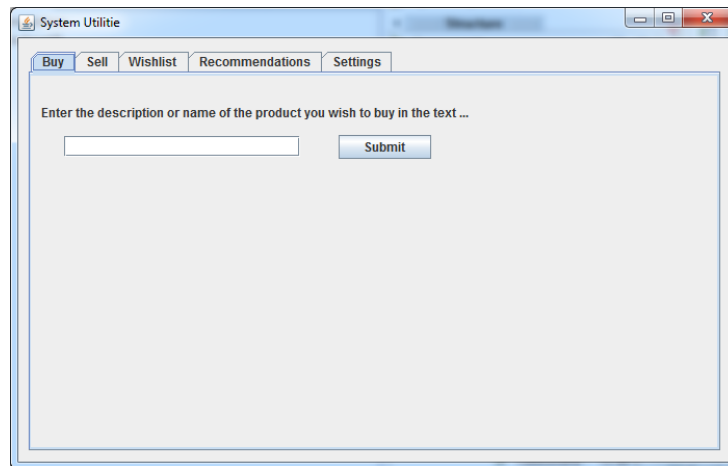
Throughout the implementation of the GUI a notation standard was followed to maintain consistency between all variable names and to improve readability, as shown in table 4.1, therefore by referring to the notation of the component the developer would not have to scroll through the entire scope to find out its type:

**Table 4.1 componenets notations used**

Component Type	Notation used / (example)
Text field	TF / (NewProductTF)
Tables	Table / (CurrentProductsT)
Buttons	B / (AddProductB)
Check Boxes	CB / (DisableNotificationCB)
Radio Buttons	RB / (FilterByNameRB)

The GUI has been implemented according to the design guidelines, where mainly one window is used containing a number of tabs, each having a particular purpose.

The *Buy* tab is where the user would be able to input the products name or details of the product they would like to purchase into the system as shown in figure 4.2:



**Figure 4.2 Implementation of the Buy tab**

Below is the backend code of the GUI shown in figure 4.2, the code contains the declaration of all UI components within this tab and it also contains an action listener method, which executes each time the *AddToBeBoughtProductB* button is pressed, upon pressing the button a DB connection to the database is established then a SQL statement is executed which adds the value within the text box as an entry to the DB, any thrown exceptions are to be handled by the try/catch block:

```
JPanel BuyingTab = new JPanel(); // Declares a new tab
tabbedPane.addTab("Buy", null, BuyingTab, null);
BuyingTab.setLayout(null); // Adds the new tab to the window

ToBeBoughtProductTF = new JTextField(); // a text field to hold the new products value
ToBeBoughtProductTF.setBounds(32, 58, 227, 20);
BuyingTab.add(ToBeBoughtProductTF);
ToBeBoughtProductTF.setColumns(10);

JButton AddToBeBoughtProductB = new JButton("Submit"); // declaring the button
AddToBeBoughtProductB.setBounds(296, 57, 89, 23);
BuyingTab.add(AddToBeBoughtProductB);

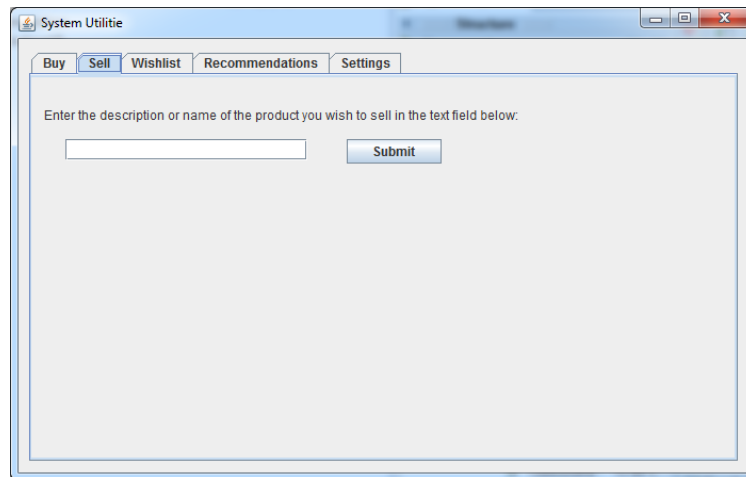
AddToBeBoughtProductB.addActionListener(new ActionListener() { // executes upon button press
    public void actionPerformed(ActionEvent arg0) {

        String stmtnt = "INSERT INTO ToBeBoughtProducts (ProductName, ProductStatus) VALUES('"+
        DBinteractions.AddToBeBoughtProducts = new DBinteractions();
        AddToBeBoughtProducts.ExecuteDBQueryNoReturn(stmtnt);

    }
});

JLabel lblNewLabel = new JLabel("Enter the description or name of the product you wish to buy");
lblNewLabel.setBounds(10, 24, 411, 23);
BuyingTab.add(lblNewLabel);
```

The *Sell* tab is where the user would be able to add the product name or details of the products would like to sell into the system as shown in figure 4.3:



**Figure 4.3 Implementation of the Sell tab**

Below is the backend code for the *sell* tab shown in figure 4.3, it is almost identical to the backend code of the *buy* tab as it carries out the same functionality:

```
JPanel SellingTab = new JPanel();
tabbedPane.addTab("Sell", null, SellingTab, null);
SellingTab.setLayout(null);

Label label = new Label("Enter the description or name of the product you wish to sell in the
label.setBounds(10, 24, 528, 23);
SellingTab.add(label);

ToBeSoldProductTF = new JTextField();
ToBeSoldProductTF.setBounds(32, 58, 227, 20);
SellingTab.add(ToBeSoldProductTF);
ToBeSoldProductTF.setColumns(10);

JButton AddToBeSoldProductB = new JButton("Submit");
AddToBeSoldProductB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        String stmtnt = "INSERT INTO ToBeSoldProducts (ProductName, ProductStatus) VALUES(+''"
        DBinteractions AddToBeSoldProducts = new DBinteractions();
        AddToBeSoldProducts.ExecuteDBQueryNoReturn(stmtnt);
    }
});
AddToBeSoldProductB.setBounds(296, 58, 89, 23);
SellingTab.add(AddToBeSoldProductB);
```

The *Wishlist* tab is used to display the current Wishlist contents and provides the user the option to add or remove items as shown in figure 4.4:

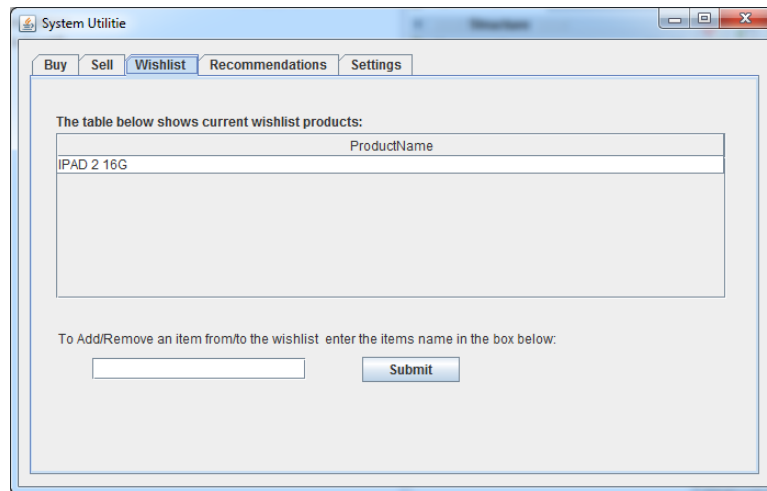


Figure 4.4 Implementation of the Wish-list tab

The code below is the backend code for the *Wishlist* tab:

```
JPanel WishlistTab = new JPanel(); // declares a new tab
tabbedPane.addTab("Wishlist", null, WishlistTab, null);
WishlistTab.setLayout(null);

Label label_1 = new Label("To Add/Remove an item from/to the wishlist enter the items ");
label_1.setBounds(22, 226, 518, 22);
WishlistTab.add(label_1);

AddRemoveWishlistProductTF = new JTextField(); // a text field to hold the product name
AddRemoveWishlistProductTF.setBounds(55, 254, 195, 20);
WishlistTab.add(AddRemoveWishlistProductTF);
AddRemoveWishlistProductTF.setColumns(10);
```

Once the Wish list tab is displayed, a table which displays the current items within the wish list is populated as indicated from the code below and shown in figure 4.4:

```
WishlistTable = new JTable(); // a table to display current wishlist items
scrollPane.setViewportView(WishlistTable); //encapsulates the table within a scroll pane to enable scrolling

String stment = "SELECT ProductName FROM WishListProducts"; //selects data from the wishlist table
DBinteractions PopulateWishlistTable = new DBinteractions(); //
WishlistTable.setModel(DbUtils.resultSetToTableModel(PopulateWishlistTable.ExecuteDBQuery(stment))
```

According to Oracles (Oracle) guidelines, each time a table object is to be instantiated the developer must define the number of columns, rows and their respective contents manually which is a time consuming task, in order to avoid this a rs2xml.jar open source library has been used, it allows the use of the *DbUtils.resultSetTabelModel()* method which automatically defines the table setting depending on contents.

An actions listener method is used to perform a number of functionality upon each press of the submit button:

- Establishes the DB connection

- Executes a SELECT all statement which retrieves all data base values.
- A while loop is used to read the values within the database one by one and assign them to a variable
- Then an if statement is used to check if the product already exists within the data base or not
- If it does not already exists, it adds it and if it does exist it deletes it!
- Break statements are used within each if-statement, to execute only once and not for the entire duration the button is pressed

```
SubmitWishlistB.addActionListener(new ActionListener() { // executes upon button press
    public void actionPerformed(ActionEvent arg0) {
        try {

            String stmtnt = "SELECT * FROM WishlistProducts WHERE ProductStatus = Yes"; //retrieves all data

            DBinteractions AddRemoveWishlistProducts = new DBinteractions();
            //ResultSet rs = AddRemoveWishlistProducts.ExecuteDBQueryMR(stmtnt);
            ResultSet rs = AddRemoveWishlistProducts.ExecuteDBQuery(stmtnt);
            String ValueInBox = AddRemoveWishlistProductTF.getText();

            while (rs.next()){ // reads every row

                String ProductName = rs.getString("ProductName"); // stores retrieved data for later process
                System.out.println(ProductName);

                if (ProductName.equals(ValueInBox) ){ // if it exists within the DB then delete it

                    stmtnt = "DELETE FROM WishlistProducts WHERE ProductName = '"+ValueInBox+"'";
                    AddRemoveWishlistProducts.ExecuteDBQueryUpdate(stmtnt);
                }
                else if (ProductName != ValueInBox){ // if it does not exist within the DB then add it

                    stmtnt = "INSERT INTO WishlistProducts (ProductName, ProductStatus) VALUES('"+ValueInBox+", Yes)";
                    AddRemoveWishlistProducts.ExecuteDBQueryNoReturn(stmtnt);
                }
            }
        } catch (SQLException e) {

            e.printStackTrace();
        }
    }
});
```

The *Recommendation* Tab displays any recommendation presented by agents and gives the user the option to filter the type of recommendation displayed through radio buttons:

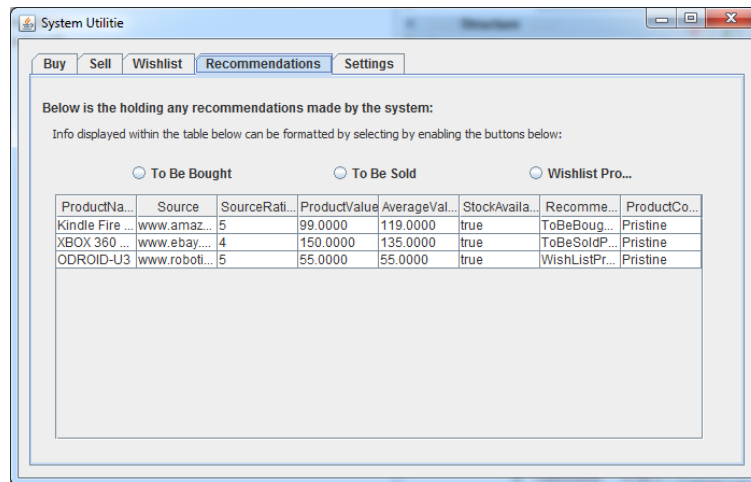


Figure 4.5 Implementation of the Recommendations tab

Once the Recommendations tab is loaded, a table will instantly populate with data from the *RecommendedProducts* table within the DB, it only displays fields that are of interest to the user as indicated from figure 4.5 and the code below:

```
table = new JTable();
scrollPane_1.setViewportViewView(table);

try {
    final Connection conn=DriverManager.getConnection(ucannaccessString);
    Statement s = conn.createStatement();
    String stmt = "SELECT ProductName, Source, SourceRating, ProductValue, AverageValue, StockAvailability,
        + "ProductCondition FROM RecommendedProducts";
    ResultSet rs = s.executeQuery(stmt);
    table.setModel(DbUtils.resultSetToTableModel(rs));
    s.close();
}

catch (Exception e)
{
    e.printStackTrace();
}
```

The tab offers a number of radio buttons which a user can interact with to filter data displayed in the table, this is implemented by using a number of action listener methods that can only be checked once at a time as indicated from the code below, upon user interaction with the radio button the table will be updated with data returned as a result of executing a SQL query:

```
ToBeBoughtRB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        WishlistProductsRB.setSelected(false);
        ToBeSoldRB.setSelected(false);

        String stmt = "SELECT ProductName, Source, SourceRating, ProductValue, AverageValue, "
            + " StockAvailability, RecommendationType, ProductCondition FROM RecommendedProducts"
            + " WHERE RecommendationType = 'ToBeBoughtProduct' ";
        DBInteractions PopulateTheRecommendedProductsTableFiltered = new DBInteractions();
        table.setModel(DbUtils.resultSetToTableModel(PopulateTheRecommendedProductsTableFiltered.ExecutedDBQuery(stmt));
    }
});
```



Functionality included within the code above is repeated for each radio button but with slight modification such as changing the SQL query.

The *Settings* Tab (figure 4.6) provides the user the option of controlling the system

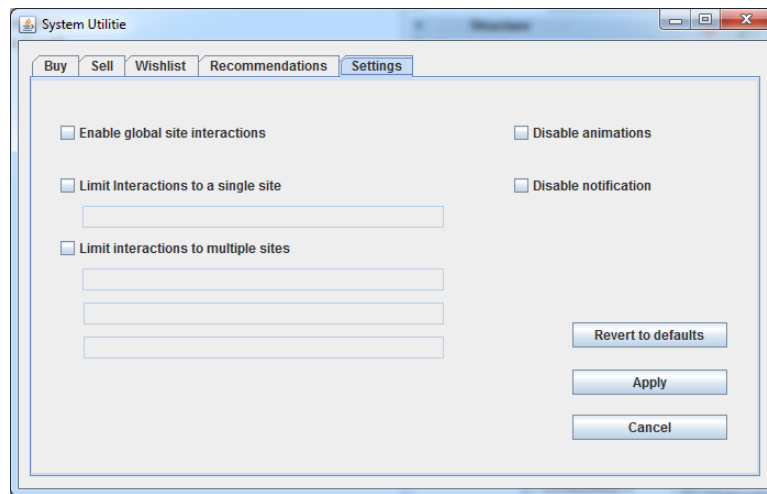


Figure 4.6 Implementation of the Settings tab

All of the above functionality has been encapsulated within a class that uses public methods offered by the *DBinteractions* class, the above functionalities are part of one function. Therefore instantiating object along with this function within main should provide the full functionalities of the GUI to the user.

### 4.2.3 The CommClass (Communications)

This class has been created to contain the functionalities that are common to perform communications between agents, the class uses the DOM standard (As discussed in the design) to interface with XML documents.

The DOM standard API uses a number of classes:

- **The document builder factory:** one of the uses of the DOM standard is to interface XML document objects to programmes, this is done by a representation of these objects in a tree structure within memory in order to do so the document builder factory is used to parse an XML document into DOM object trees, allowing for reference to specific contents of the file.

The code below demonstrates the use of this class:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

- **The document builder:** the instantiation of the document builder factory allows for the instantiation of the document builder class, this class allows XML documents to be

parsed from a variety of resources, the one used below is from URL local to the workstation:

```
private static File f = new File("C:/Users/Aboudi's/Desktop/ToBuyingAgent.xml");
DocumentBuilder builder = factory.newDocumentBuilder();
```

- **The node list:** according to the DOM standard, everything within an XML document is considered to be a node, therefore the node list (memory structure similar to an array) retrieves all elements and tags within the XML documents and allows the developer to deal with them by referring to indexes or element names, the code below instantiates this class and assigns its object values the value of a particular element from the previously declared document object that parses an XML document into a DOM document object:

```
Document doc = builder.parse(f); // "C:/Users/Aboudi's/C
NodeList list = doc.getElementsByTagName("CommStatus");
```

The number of elements within the *nodelist* can be accessed by the *getlength* method, the node list can be accessed similarly to an array which is via for loops:

```
for (int i = 0; i < list.getLength(); i++) {
    // do something with the elements
    Element item = (Element)list.item(i);
    item.getFirstChild().setTextContent("Read"); // give it a text value
}
```

There are also a number of other classes that are used to interface applications to files, as shown below:

- **Serialize:** converts document data into a sequence of bytes to be written into a file.
- **Output format:** to set the format configurations of an XML document.
- **Output stream:** used to allow writing streams to files

These are the main functionalities and features of the DOM standard that will be used to implement communications between agents.

The functionalities included within this class are common to both agents and both methods of communication (negotiation/ message exchange) and are derived from the communications design:

- A function that deletes files when read
- A function that creates a file (compliant with the design and protocol created)
- A function that modifies the files contents to indicate that a message has been read

The following function takes in a single parameter of type File, this parameter will be the file path that the agent is to deal with, as mentioned within the design stage that a file will be deleted

by the agent who sent it once the receiving agent has read it, reading the file is indicated by the `commStatus` tag as it would either have a value of `read` or `unread`.

The sending agent will use the function below to check whether the value within that specific tag is changed to `read` or not, if it has then it deletes it:

```
public void DeleteFileWhenRead(File f) throws ParserConfigurationException // This Function
, SAXException, IOException
{
    String MessageStatus = "";
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

    DocumentBuilder builder = factory.newDocumentBuilder();
    org.w3c.dom.Document doc = builder.parse(f); // remember this should be the file path

    NodeList list = doc.getElementsByTagName("CommStatus");

    System.out.println(list.getLength());

    for (int i = 0; i < list.getLength(); i++) {
        Element item = (Element)list.item(i);
        MessageStatus = (String) item.getFirstChild().getNodeValue();
    }

    if(MessageStatus.equals("Read"))
    {
        f.delete();
    }
}
```

The following function is used to create an XML file, this will be used by both agents each time they need to carry out some form communication in between them.

The function takes in two parameters, the first is the product name this will be the value store within the *MessageValue* tag, the other parameter is the name of the receiving agent the one which the message is targeted for.

The files contains three XML tags (elements) each has been declared and assigned a value as shown in the code below, each one of those elements has a child element which is the text value (which explains the `append child` method), after values have been assigned to each tag, these tags are to be placed in a file, this is done by instantiating a file object and defining the path of the file, note that the name of the file will depend on the *receptionist* parameter as mentioned within the design it is either going to be *ToSellingAgent* or *ToBuyingAgent*:

```

//To send in the name of the product to be recommended and the stated receptionist
public void CreateFile (String ProductName, String Receptionist) throws ParserConfi

    try {
        DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
        Document xmlDoc = docBuilder.newDocument();

        Text ProductNameToF = xmlDoc.createTextNode(ProductName);
        Text CommType = xmlDoc.createTextNode("MessageExchange");
        Text ReadStatus = xmlDoc.createTextNode("Unread");

        Element rootElement = xmlDoc.createElement("CommType");
        rootElement.appendChild(CommType);

        Element Status = xmlDoc.createElement("CommStatus");
        Status.appendChild(ReadStatus);

        Element Value = xmlDoc.createElement("CommValue");
        Value.appendChild(ProductNameToF); // becomes the child of the tag Value

        rootElement.appendChild(Status); // becomes the comm type of the tag Value
        rootElement.appendChild(Value); // becomes the commtype of the tag Value

        xmlDoc.appendChild(rootElement); // adds the root elements to the document
        OutputFormat outFormat = new OutputFormat(xmlDoc);

        File xmlFile = new File("C:/Users/Aboudi's/Desktop/"+Receptionist+".xml");
        FileOutputStream outputStream = new FileOutputStream(xmlFile);
        XMLSerializer serializer = new XMLSerializer(outputStream,outFormat);
        serializer.serialize(xmlDoc);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

Upon the creation of the XML document some tags will have a default value that may be changed after it has been created, which explains why the value of some tags is hard coded within the above function.

The below function modifies the contents of the xml file, this is done by both agents, it gets called once the receiving agent has read and acted upon a message, doing so means that the *CommStatus* tag has to be changed to read.

The function take in a single parameter which is the file path of the file the agent is dealing with, the function first checks if the file exists or not, if it does then it identifies the tag it wants to deal with and then assigns to it a new value:

```

// requires the file name of the file that should be modified passed to it
public void MarkMessageAsRead(File f) throws IOException, SAXException, ParserConfigException {
    if(f.exists() && !f.isDirectory()) {
        //System.out.println("the file exists");

        String MessageStatus="";
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse(f); // "C:/Users/Aboudi's/Desktop/ToSellingAgent.xml"

        NodeList list = doc.getElementsByTagName("CommStatus");

        //System.out.println(list.getLength());

        for (int i = 0; i < list.getLength(); i++) {
            Element item = (Element)list.item(i);
            //System.out.print(((Node) item).getFirstChild().getNodeValue());
            MessageStatus = (((Node) item).getFirstChild().getNodeValue());
            item.getFirstChild().setTextContent("Read");

            OutputFormat outFormat = new OutputFormat(doc);
            FileOutputStream outputStream = new FileOutputStream(f);
            XMLSerializer serializer = new XMLSerializer(outputStream,outFormat);
            serializer.serialize(doc);
        }
    }
    else {
        System.out.println("does not exist exists");
    }
}
}

```

#### 4.2.4 The message exchange class

As mentioned within the design stage that there are two types of communication which occurs between agents(negotiation and message exchange), it is also mentioned that functionalities specific to each form of communication will be packaged within a class, therefore this class packages the functionalities required to implement message exchange between the two agents.

This class contains one functionality that uses the functions created within the *CommClass* except it adds a few more behaviours and conditions to using those functions, the function is called *MonitorReadRespondModify*:

- The function reads and responds to file contents and acts upon them
- It checks if it should delete the file
- It checks if it exists
- Loads the file contents into memory

The function contains two parameters, the file path of the file which the agent should deal with and a string containing the name of the agent:

The agents normally have to deal with messages in two methods, when they are sent and when they are received, the agents monitor the status of the message sent and await until its read and

then they delete it, the other method is when they are received in such event the agent must read the message and act upon it and then change its status to read.

In order to do so the agent must confirm its identity to know which message to read and which to monitor and then delete, in order to do this the fact that the receiving agents name is mentioned in the file name of the message and the string parameter of the function can be taken advantage of

- Therefore the first if statement checks whether its name is mentioned within the file name or, if it is not mentioned then it identifies that it is the agent who sent it, therefore it waits until it is read then deletes it.
- Otherwise if the message contains its name then that means that the agent should read it, therefore it loads the message contents into memory.
- Then it checks if the message has already been read by it or not, if it has been read then it just ignores it.
- If it has not been read by it then it acts upon its contents by adding the *CommValue* into the database and marking the message as read (both done by using the methods previously created in other classes).

```
public class MessageExchange {

    /*
     * This class should inherit from the CommsClass
     * 1. A function that reads and responds to file contents
     *     - Checks if it should delete it
     *     - Checks if it exists
     *     - Gets the tag contents
     *     - Acts upon contents
     */
    //File f = new File("C:/Users/Aboudi's/Desktop/ToSellingAgent.xml");

    //parameters(name and directory of the file you want the agent to read,the name of this agent)
    public void MonitorReadRespondModify(File f, String AgentName) throws ParserConfigurationException,

    if(!f.getName().contains(AgentName)){ // if it does not contain the name then it must be the sender,
        // then check if the file is read, if its is delete if it is not then do nothing
        CommClass CheckIfItIsRead = new CommClass();
        CheckIfItIsRead.DeleteFileWhenRead(f);
    }
    else if(f.exists() && !f.isDirectory()) {

        String ProductNameInFile=""; // to hold the value stored within the value tag
        String CommStatusInFile=""; // to hold the value stored within the CommStatus tag

        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse(f); // "C:/Users/Aboudi's/Desktop/ToSellingAgent.xml" //not needed

        NodeList list = doc.getElementsByTagName("CommValue");
        NodeList list2 = doc.getElementsByTagName("CommStatus");

        for (int i = 0; i < list.getLength(); i++) {
            Element item = (Element)list.item(i);
            Element item2 = (Element)list2.item(i);
```

```

//System.out.print(((Node) item).getFirstChild().getNodeValue());
ProductNameInFile = (((Node) item).getFirstChild().getNodeValue());
CommStatusInFile=(((Node) item2).getFirstChild().getNodeValue());

String SalesTable = "ToBeSoldProducts";
String BuyingTable="ToBeBoughtProducts";
String VariableNeutral = "";

if(f.getName().contains("Buying")){
    VariableNeutral = SalesTable;
}else if(f.getName().contains("Buying")){
    VariableNeutral = BuyingTable;
}

if(CommStatusInFile.equals("Read")){ // if the tag within the file is read then dont add it
    break;
}
else{ //// if the tag within the file is not read then add it
//String stmt40 = "INSERT INTO "+VariableNeutral+" "+" "+" (ProductName, ProductStatus) VALUES ('"+Produ
String stmt40 = "INSERT INTO ToBeSoldProducts (ProductName, ProductStatus) VALUES ('"+ProductNameInFile+
DBinteractions UpdateFileContentsToDB = new DBinteractions();
UpdateFileContentsToDB.ExecuteDBQueryNoReturn(stmt40);
// then change the tag value by calling the modify

        CommClass ModifyIt = new CommClass();
        ModifyIt.MarkMessageAsRead(f);
    }
}
else {
    System.out.println("does not exist exists");
}
}
}

```

The implementation of this class demonstrates the coordination between agents (as one agents behaves upon the action of another i.e. deleting messages when they are read), it also demonstrates the communication aspect of MAS and the benefits of such capability.

## 4.2.5 The Agents Class

This class will contain the functionalities that are common to, and used by both agents, the class may use functions that are created by other classes, such as the DB interaction class.

For now this class contains one functionality, common between the two agents, further functionalities can be added to the finished product.

The following functionality calculates the average value of each product, it begins by executing a SQL statement that retrieves the product name of every product within the *ToBeBoughtProducts* table only if the status of such entry is true otherwise the agents will ignore it:

- The result is stored within a *ResultSet* variable, as multiple results are returned then a while loop can be used to process the result each time it returns, this loop retrieves the value stored within the product name column and stores it within a variable
- Another SQL statement uses this variable to retrieve the value of this specific product from the product details table which holds multiple entries with the same product name, therefore the statement retrieves the value for each entry, another embedded while loop is used to read and process each value returned, as mentioned within the design, the

values are added to an array then a counter increments each time a value is added and the cumulative of all values would be known therefore the average can be calculated:

Advantage of embedded loops: the property of embedded loops have been advantageous on this occasion, as while loops normally execute the outer loop once each time the inner loop is has finished iterating

In this scenario the outer loop retrieves the product name of which the average is to be calculated, then the inner loop retrieves the values required to calculate the average of the product within the outer loop, this is a repeating process, eventually calculating the average value for every single product.

```
public class Agents {  
  
    /*  
     * 1. calculates the average of a product value of a set of products  
     */  
  
    //The agent which the average is going to be calculated for  
    public void CalculateAverageValue() throws SQLException{ // calculates the average for each product in the DB  
  
        DBInteractions Execute = new DBInteractions(); // a new instance to allow for use of methods  
  
        String stmtnt = "SELECT ProductName FROM ToBeBoughtProducts WHERE ProductStatus = 'True'";  
        ResultSet rs = Execute.ExecuteDBQuery(stmtnt); // returns and store the results  
  
        float[] Value = new float[10]; // to store the  
        float AverageValue=0; // to hold the average value  
        float cumulative = 0; // stores the cumulative of value  
        int i=0; // incrementer to keep track of values  
  
        while(rs.next()){  
            String ProductName = rs.getString("ProductName");  
            //System.out.println(ProductName); // it works so far  
  
            String stmtnt2 = "SELECT ProductValue From ToBeBoughtProductDetails WHERE ProductName = '"+ProductName+"'"; // "+ProductName+"WII U WIFI  
            ResultSet rs2 = Execute.ExecuteDBQuery(stmtnt2);  
  
            while(rs2.next()){  
                float ProductValue = rs2.getFloat("ProductValue");  
                //System.out.println("ProductValue var = " +ProductValue);  
                Value[i] = ProductValue;  
                cumulative = cumulative+ Value[i];  
                ++i;  
                AverageValue= cumulative/i;  
                //System.out.println("AverageValue = "+AverageValue);  
                //System.out.println(i);  
            }  
        }  
    }  
}
```

Note that the function created above is not related to a specific product rather it deals with all products, this is done intentionally therefore the agents can simply use this function as an algorithm to calculate average values and act upon the result

## 4.2.6 The SellingAgent Class

This class does not contain much functionality rather it just calls the MonitorReadModify function from the message exchange class



```

public class TheSellingAgent {

    private MessageExchange HandleMessages = new MessageExchange();
    private static File f = new File("C:/Users/Aboudi's/Desktop/ToSellingAgent.xml");

    public void InitialiseAndLaunch() throws ParserConfigurationException, SAXException, IOException{

        HandleMessages.MonitorReadRespondModify(f, "SellingAgent");

    }

}

```

#### 4.2.7 The BuyingAgent class

The buying agent class represents the buying agent, therefore all functionalities contained within it are specific to the agent, it uses the property of embedded while loops to its advantage in the same manner it was used in the Agents class, it also uses a number of functionalities which have been created in other classes.

The functionalities contained within this class are derived from the agents goals and roles:

- It calculate the average value(sales value) of a product
- It compares products within the database and performs a recommendation accordingly
- It creates a message file and sends it to the selling agent upon each recommendations
- Deals with messages sent to it.

All the above functionality are to be performed automatically once the agent is instantiated.

First the agent calls the *MonitorReadModifyRespond* method to deal with any files sent to it, the function has been discussed previously within the Message exchange class therefore it will not be discussed here:

```

HandleMessages.MonitorReadRespondModify(f, "BuyingAgent");
private MessageExchange HandleMessages = new MessageExchange();

```

The class contains another private function called the Perform recommendation, the code for this function is explained in sequential steps, the code that implements this function is also shown below:

1. The function executes a SQL statement that retrieves all product values which have the status of true from the *ToBeBoughtProducts* table, the result returned by execution is stored within a Result Set variable.
2. The result set variable is iterated through by the use of a while loop, iterations occur each time a result is returned.
3. Each time a result is returned, it gets stored within a variable, the stored value is the product name returned by the SQL query execution.
4. This product name is used to retrieve all product details of the same product name from within the *ToBeBoughProductsDetails* one by one.

5. Another while loop is used to iterate through the values returned by the second statement.
6. Each time the loop executes, it extracts data from the SQL query execution result, the data extracted gets stored within variables.
7. These variables are used to check if the product within the table meets the minimum requirements through the use of an if-statement.
8. If the product does not meet the minimum requirements/criteria to make a recommendation then the loop iterates again retrieving a new set of results.
9. If the product meets the minimum criteria then a SQL statement is used to retrieve the details of the current recommended product.
10. This is done to compare the product that meets the minimum recommendation criteria to the one that already has been recommended, therefore if the product is better then the current recommended product then it takes its position, if not then the loop keeps iterating.

These function should iterate through each value within the database and carry out comparisons for each product to identify whether it is suitable to recommend or not.

```
private void PerfromRecommendation() throws ParserConfigurationException, IOException{

    DBinteractions Execute = new DBinteractions();
    try{
        String stmtnt = "SELECT ProductName FROM ToBeBoughtProducts WHERE ProductStatus = 'True'";
        //ResultSet rs = s.executeQuery(stmtnt); // execute a statement without returning anything
        ResultSet rs= Execute.ExecuteDBQuery(stmtnt);

        while(rs.next()) {
            String ProductName = rs.getString("ProductName");

            String stmtnt2 ="SELECT * From ToBeBoughtProductDetails WHERE ProductName = '"+ProductName+"'"; //"+ProductName+"WIII U WIF:
            ResultSet rs3= Execute.ExecuteDBQuery(stmtnt2);

            String Source;
            float SourceRating;
            float ProductValue;
            boolean DeliveryAvailability;
            boolean StockAvailability;

            String ProductNameInRT;
            float SourceRatingInRT;
            float ProductValueInRT;
            boolean DeliveryAvailabilityInRT;
            boolean StockAvailabilityInRT;

            while(rs3.next()){
                // these are the values from the ProductDetailsTable
                Source= rs3.getString("Source");
                SourceRating= rs3.getFloat("SourceRating");
                ProductValue= rs3.getFloat("ProductValue");
                DeliveryAvailability= rs3.getBoolean("DeliveryAvailability");
                StockAvailability= rs3.getBoolean("StockAvailability");

                //if((ProductValue <= AverageValue )&&( SourceRating>= 2.5)&&(DeliveryAvailability=true)&&(StockAvailability=true)){
                if(( SourceRating>= 2.5)&&(DeliveryAvailability=true)&&(StockAvailability=true)){ // defining the recommendation cr:
                    String stmtnt3 ="SELECT * From RecommendedProducts WHERE ProductName = '"+ProductName+"' AND RecommendationType='
                    ResultSet rs4 = Execute.ExecuteDBQuery(stmtnt3);

                    while(rs4.next()){

                        ProductNameInRT= rs4.getString("ProductName");
                        SourceRatingInRT= rs4.getFloat("SourceRating");
                        ProductValueInRT= rs4.getFloat("ProductValue");
                        DeliveryAvailabilityInRT= rs4.getBoolean("DeliveryAvailability");
                        StockAvailabilityInRT= rs4.getBoolean("StockAvailability");
```



```

public class AgentSys {

    /**
     * Launch the application.
     * @throws ParserConfigurationException
     * @throws IOException
     * @throws SAXException
     */

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {

                    //////////Initialises and loads the GUI//////////
                    UIclass InitializeUI = new UIclass();
                    InitializeUI.AgentsGUI();
                    //////////

                    //////////Initialises and launches the selling Agent////////
                    TheSellingAgent SellingAgent = new TheSellingAgent();
                    SellingAgent.InitialiseAndLaunch();
                    //////////

                    //////////Initialises and launches the selling Agent////////
                    TheBuyingAgent BuyingAgent = new TheBuyingAgent();
                    BuyingAgent.InitialiseAndLaunch();
                    //////////

                    //////////everything above this point should not be changed

                }
                catch(Exception e ){
                    e.printStackTrace();
                }
            }
        });
    }
}

```

# Chapter 5 Testing

The O-MaSE agent methodology used at earlier stages does not specify any testing tasks, however this is not an issue, as there is a number of existing useful testing methodologies which can be made use of. The methodologies used throughout this stage will be White and Black box testing.

## 5.1 White box testing

This method is concerned with testing each functionality or part of the system individually.

**Function testing:** which is similar to the statement testing technique offered by this methodology. As an object oriented approach was used to implement the system then it is possible to use this technique by testing the main functions of each class against their expected outcome:

**The *DBinteractions* class:** contains the following functionality:

A function that executes a SQL statement and returns a result:

```
public ResultSet ExecuteDBQuery(String SQLst){
```

A function that executes a SQL statement without returning a result:

```
public void ExecuteDBQueryNoReturn(String SQLst){
```

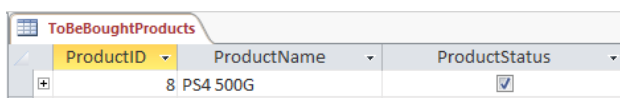
A function that executes an update or delete SQL statement:

```
public void ExecuteDBQueryUpdate(String SQLst){
```

The functions are then each invoked individually:

The function *ExecuteDBQuery* has been tested through the statements below which executes a SQL statement that returns values from the database, the outcome was successful as the results were displayed in the console window in figure 5.1 as they were stored in the DB figure 5.2

```
DBinteractions Testing = new DBinteractions();  
String TestStmnt = "SELECT * From ToBeBoughtProducts";  
ResultSet rs = Testing.ExecuteDBQuery(TestStmnt);  
while(rs.next()){  
    System.out.println(rs.getInt("ProductID") + " " + rs.getString("ProductName") + " " + rs.getBoolean("ProductStatus"));  
}
```



ProductID	ProductName	ProductStatus
8	PS4 500G	<input checked="" type="checkbox"/>
9	WII U WIFI	<input checked="" type="checkbox"/>

Figure 5.2 Values stored within the DB

```
8 PS4 500G true  
9 WII U WIFI true
```

Figure 5.1 Values read from the DB

The function *ExecuteDBQueryNoReturn* has been tested by using the statements below, which execute a SQL statement that adds a new product to the DB as expected upon executing the statement a new product was added to the DB as shown in figure 5.3

```
DBinteractions Testing = new DBinteractions();

String TestStmnt = "INSERT INTO ToBeBoughtProducts (ProductName, ProductStatus) VALUES ('Headphones', 'True')";
Testing.ExecuteDBQueryNoReturn(TestStmnt);
```



ProductID	ProductName	ProductStatus
8	PS4 500G	<input checked="" type="checkbox"/>
9	WII U WIFI	<input checked="" type="checkbox"/>
13	Headphones	<input checked="" type="checkbox"/>

Figure 5.3 Vlaues stored successful within the DB

The function *ExecuteDBQueryUpdate* has been tested through the statements below which delete a row from the database, the values within the database before deletion are shown in figure 5.5 and after in figure 5.4:

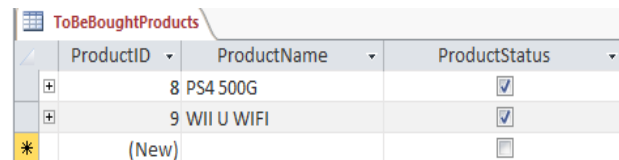
```
DBinteractions Testing = new DBinteractions();

String TestStmnt = "DELETE FROM ToBeBoughtProducts WHERE ProductName = 'Headphones' ";
Testing.ExecuteDBQueryUpdate(TestStmnt);
```



ProductID	ProductName	ProductStatus
8	PS4 500G	<input checked="" type="checkbox"/>
9	WII U WIFI	<input checked="" type="checkbox"/>
13	Headphones	<input checked="" type="checkbox"/>

Figure 5.5 DB before deleting values



ProductID	ProductName	ProductStatus
8	PS4 500G	<input checked="" type="checkbox"/>
9	WII U WIFI	<input checked="" type="checkbox"/>
*	(New)	<input type="checkbox"/>

Figure 5.4 DB after deleting values

**The *Uiclass*:** the method in which this class will be tested will slightly differ from the routine followed throughout the use of the white box testing method, as testing will be carried out by interacting with each tab separately and verifying the outcome of each interaction.

The *Buy* tab: if a value is added to the text field within this tab then the submit button is pressed, then this should cause the product within the text field to be added to the DB, the outcome of this test was successful as shown in figure 5.6 and 5.7:

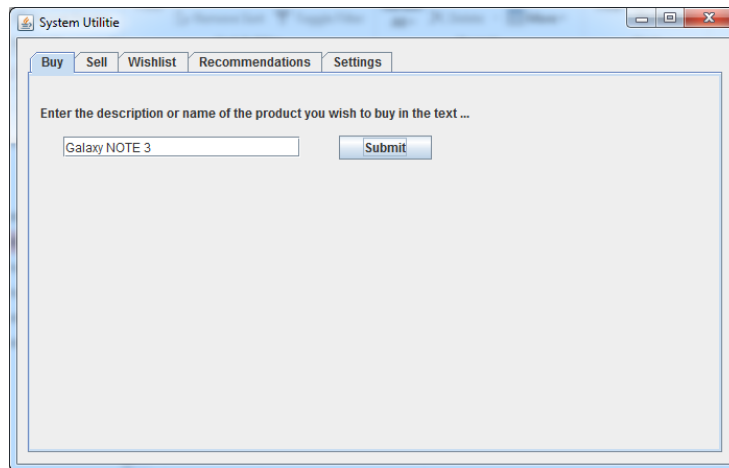


Figure 5.6 Testing the Buy tab

WishListProducts		ToBeBoughtProducts	
ProductID	ProductName	ProductStatus	
8	PS4 500G	<input checked="" type="checkbox"/>	
9	WII U WIFI	<input checked="" type="checkbox"/>	
14	Galaxy NOTE 3	<input checked="" type="checkbox"/>	

Figure 5.7 the product was added to the DB as expected

The *Sell* tab: has the same functionality as the *Buy* tab except products are added to the *ToBeSoldProducts* within the DB, the outcome of this test was successful as shown in figure 5.8 and 5.9:

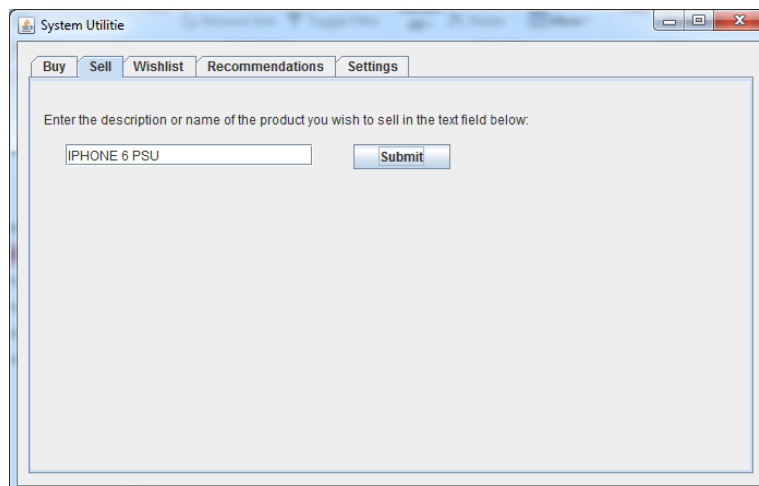
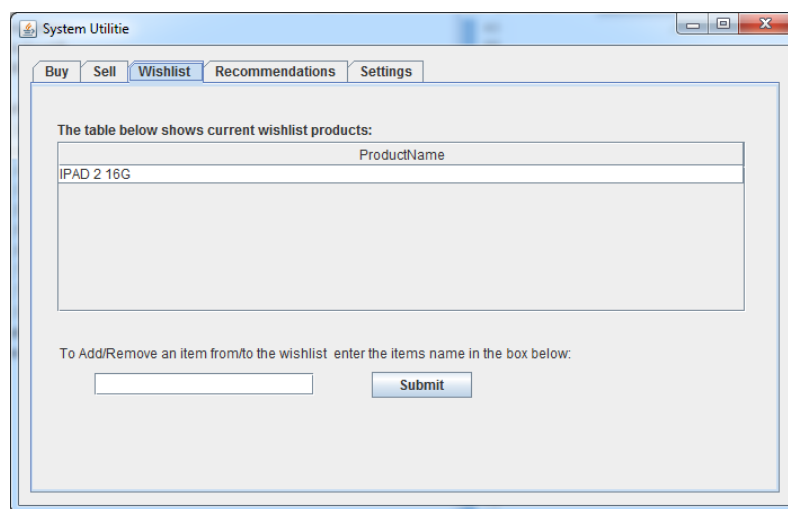


Figure 5.8 Testing the sell tab by adding a product

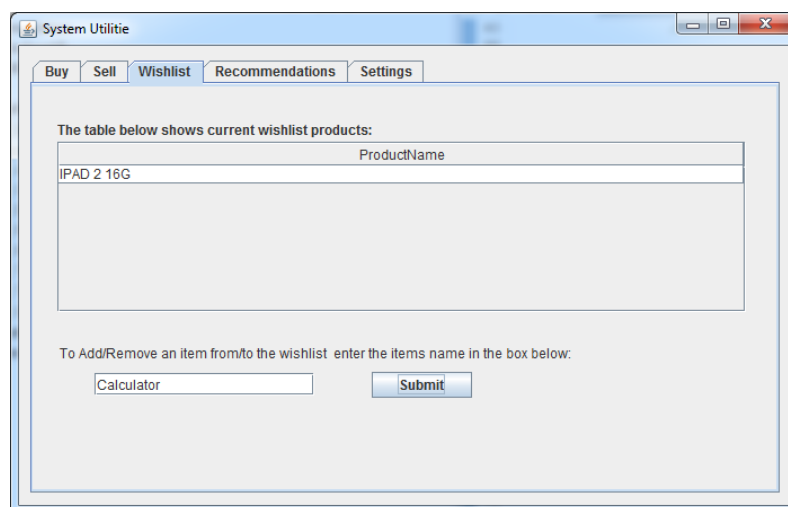
WishListProducts	ToBeBoughtProducts	ToBeSoldProducts
ProductID	ProductName	ProductStat
86	ASUS TAB6 7G	<input checked="" type="checkbox"/>
87	WII U WIFI	<input checked="" type="checkbox"/>
88	IPHONE 6 PSU	<input checked="" type="checkbox"/>

**Figure 5.9** test outcome was successful as the product was added to the DB

The *Wishlist* tab: the *Wishlist* tab should populate a table containing products that are stored within Wish list each time the tab is loaded, as expected such test was successful as shown in figure 5.10. The *Wishlist* tab offers the user the option to add or remove products to/from the Wish list, this has been tested where a new product is added figure 5.11 & 5.12 then deleted figure 5.13



**Figure 5.10** The wish list tab displays all values upon startup



**Figure 5.11** testing the wish list tab by adding a product



WishListProducts		
ProductID	ProductNan	ProductStat
1	IPAD 2 16G	<input checked="" type="checkbox"/>
117	Calculator	<input checked="" type="checkbox"/>

Figure 5.13 A product was added successfully

ToBeSoldProducts		
ProductID	ProductNan	ProductStat
1	IPAD 2 16G	<input checked="" type="checkbox"/>
(New)		<input type="checkbox"/>

Figure 5.12 the product was removed successfully

The *Recommendations* tab: this tab should populate a table containing all recommendations made by agents and their related info, each time the tab loads. This has been tested as shown in figure 5.14 the tab contains filtering options as well these have been tested as shown in figure 5.15 – 5.17

ProductNa...	Source	SourceRati...	ProductValue	AverageVal...	StockAvaila...	Recomm...	ProductCo...
Kindle Fire ...	www.amaz...	5	99.0000	119.0000	true	ToBeBoug...	Pristine
XBOX 360 ...	www.ebay...	4	150.0000	135.0000	true	ToBeSoldP...	Pristine
ODROID-U3	www.roboti...	5	55.0000	55.0000	true	WishListPr...	Pristine
Wii U WIFI	www.game...	5	320.0000	0.0000	true	ToBeBoug...	

Figure 5.14 The recommendations tab loads all recommendations as expected

ProductNa...	Source	SourceRati...	ProductValue	AverageVal...	StockAvaila...	Recomm...	ProductCo...
Kindle Fire ...	www.amaz...	5	99.0000	119.0000	true	ToBeBoug...	Pristine
Wii U WIFI	www.game...	5	320.0000	0.0000	true	ToBeBoug...	

Figure 5.15 Filtering options work as expected

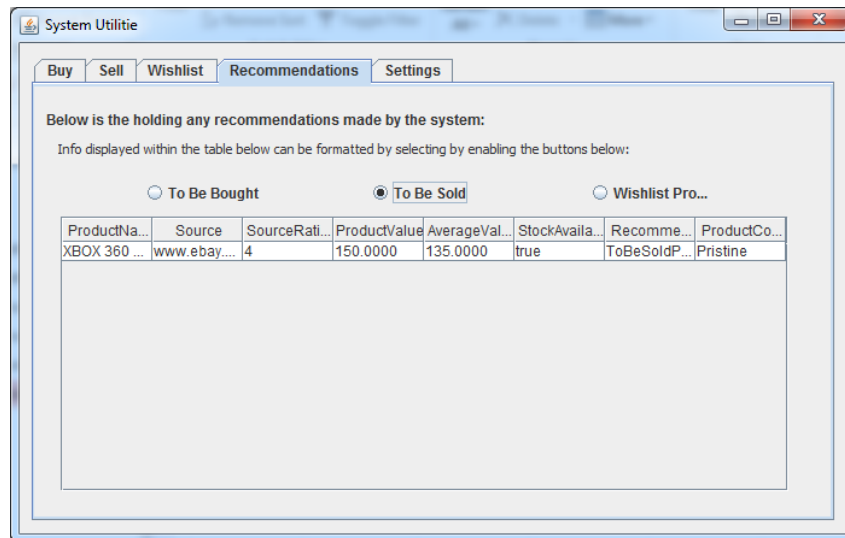


Figure 5.16 Filtering options work as expected

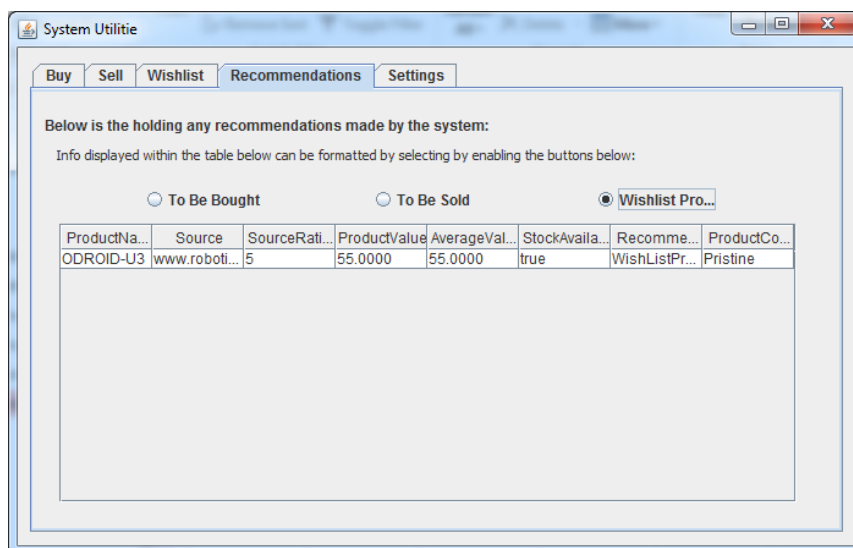


Figure 5.17 Filtering options work as expected

The **CommClass**: has the following functionalities:

A function to delete a file when it is read

```
public void DeleteFileWhenRead(File f) throws ParserConfigurationException
```

A function that modifies the contents of the file to mark it as *Read*:

```
public void MarkMessageAsRead(File f) throws IOException, SAXException, ParserConfigurationException{
```

A function that creates a file:

```
public void CreateFile (String ProductName, String Receptienest) throws ParserConfigurationException, IOException{
```

Each one of these functionalities has been tested individually:

The delete file function has been tested through the code below which deals with a file that has a message status tag with the text value of *Read* figure 5.18 as expected as a result the file was deleted:

```
File f = new File("C:/Users/Aboudi's/Desktop/ToSellingAgent.xml");
CommClass Test = new CommClass();
Test.DeleteFileWhenRead(f);
```

```
<?xml version="1.0" encoding="UTF-8"?>
<CommType>
  MessageExchange
    <CommStatus>Read</CommStatus>
    <CommValue>WII U WIFI</CommValue>
  </CommType>
```

Figure 5.18 The file which has the status of read prior to deletion

The mark message as read function has been tested through the code below, it specifies the file to be modified then it changes the contents of the message status tag to read, figure 5.29 and 5.20 show the file before and after execution of the code:

```
File f = new File("C:/Users/Aboudi's/Desktop/ToSellingAgent.xml");
CommClass Test = new CommClass();
Test.MarkMessageAsRead(f);
```

```
<?xml version="1.0" encoding="UTF-8"?>
<CommType>
  MessageExchange
    <CommStatus>Unread</CommStatus>
    <CommValue>Cisco I27</CommValue>
  </CommType>
```

Figure 5.19 Before function execution

```
<?xml version="1.0" encoding="UTF-8"?>
<CommType>
  MessageExchange
    <CommStatus>Read</CommStatus>
    <CommValue>Cisco I27</CommValue>
  </CommType>
```

Figure 5.20 After function execution

The *CreateFile* function has been tested through the statements below, as a result of executing those statements, a file was created in the expected directory with the expected contents as shown in figure 5.21:

```
CommClass Test = new CommClass();
Test.CreateFile("Cisco I27" , "ToSellingAgent");
```

```
<?xml version="1.0" encoding="UTF-8"?>
<CommType>
  MessageExchange
    <CommStatus>Unread</CommStatus>
    <CommValue>Cisco I27</CommValue>
  </CommType>
```

Figure 5.21 A file has been created as expected

The **MessageExchange** class: contains a single function that checks if it should delete a file, it acts upon the tag contents, modifies files and creates new ones.

```
public void MonitorReadRespondModify(File f, String AgentName)
```

In order to test this class, the XML file shown in figure 5.22 has been created, then the function is called with the parameters of the file path, and the agent name of the agent using the function:

```
File f = new File("C:/Users/Aboudi's/Desktop/ToSellingAgent.xml");
MessageExchange Test = new MessageExchange();
Test.MonitorReadRespondModify(f, "BuyingAgent");
```

```
<?xml version="1.0" encoding="UTF-8"?>
<CommType>
  MessageExchange
  <CommStatus>Unread</CommStatus>
  <CommValue>Word 365</CommValue>
</CommType>
```

Figure 5.22 The file to be dealt with by the agent

First the function checks if the file has not already been read, if it has not been read then it adds the text within the *CommValue* tag into the *ToBeBoughtProducts* table within the DB, as the file in figure 5.22 a status of *Unread* then the function will add its contents into the DB, as shown in figure 5.23:

ToBeBoughtProducts			
	ProductID	ProductName	ProductStatus
+	8	PS4 500G	<input checked="" type="checkbox"/>
+	9	WII U WIFI	<input checked="" type="checkbox"/>
+	14	Galaxy NOTE 3	<input checked="" type="checkbox"/>
+	15	Word 365	<input checked="" type="checkbox"/>

Figure 5.23 File contents added to the DB as expected

After the function has acted upon the contents of the XML file, it should mark it as read, which is implemented successfully as shown in figure 5.24:

```
<?xml version="1.0" encoding="UTF-8"?>
<CommType>
  MessageExchange
  <CommStatus>Read</CommStatus>
  <CommValue>Word 365</CommValue>
</CommType>
```

Figure 5.24 File status changed after execution of the function as expected

The **Agents** class: currently only contains a single functionality, which calculates the average of every single product based on price data collected from different sites and stored within the *ToBeBoughtProductDetails* table.

```
public void CalculateAverageValue() throws SQLException{ // calculates the average for each product in the DB
```

The function has been called as shown below:

```
Agents Test = new Agents();
Test.CalculateAverageValue();
```

The data which the function will base its average calculations upon is as shown in figure 5.25:

ProductID	ProductName	Source	SourceRating	ProductValue	DeliveryAvailab	StockAvailabili
2	WII U WIFI	www.game.co	5	£320.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	WII U WIFI	www.game.co	5	£230.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
4	PS4 500G	www.game.co	5	£220.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
5	PS4 500G	www.ebay.cor	3	£440.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 5.25 DB values to be used for average price calculations

A few print out statements have been used within the function to print out the results on the console window

```
AverageValue of PS4 500G is 330.0
AverageValue of WII U WIFI is 302.5
```

Figure 5.26 an average price of each product was calculated successfully

**The *BuyingAgent* class:** the buying agent makes use of the functions previously created within other class and tested, therefore the all functions that are included within this class will not be tested individually, as they have been tested before however the behaviour of this class will be tested during the black box testing method.

There is one function which is special to this class and has not been tested, therefore it will now be tested:

```
private void PerfromRecomendation() throws ParserConfigurationException, IOException{
```

The purpose of the function is to make recommendation based on a comparison of product within the data base, the recommendation will be made if the product info within the *ProductDetails* Table is better than the current recommended product.

As shown in figure 5.27 the current recommended product has the product value of £320.00 which means if a new entry is added to the *ProductDetails* Table with a lower value (Figure 5.29), it should replace the current recommendation.

Upon launching the system the agent should update the *RecommendationTable* with the better entry, this test performed successfully as shown in figure 5.30

ProductID	ProductName	Source	SourceRating	ProductValue	DeliveryAvai	StockAvaila	AverageValue	RecommendationStatus	ProductConditio	RecommendationType
1	Kindle Fire HD	www.amazon.com	5	£99.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	£119.00	<input checked="" type="checkbox"/>	Pristine	ToBeBoughtProduct
2	XBOX 360 50G	www.ebay.com	4	£150.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	£135.00	<input checked="" type="checkbox"/>	Pristine	ToBeSoldProduct
3	ODROID-U3	www.robotics.com	5	£55.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	£55.00	<input checked="" type="checkbox"/>	Pristine	WishListProduct
18	WII U WIFI	www.game.com	5	£320.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	£0.00	<input type="checkbox"/>	Pristine	ToBeBoughtProduct

Figure 5.27 Current recommended products

ProductID	ProductName	Source	SourceRating	ProductValue	DeliveryAvailab	StockAvailabili
2	WII U WIFI	www.game.co	5	£320.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	WII U WIFI	www.game.co	5	£230.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
4	PS4 500G	www.game.co	5	£220.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
5	PS4 500G	www.ebay.cor	3	£440.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
6	WII U WIFI	www.uk-deal	5	£200.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 5.28 A new product is added to the ToBeBoughtProductDetails Table

ProductID	ProductName	Source	SourceRating	ProductValue	DeliveryAvai	StockAvaila	AverageValue	RecommendationStatus	ProductConditio	RecommendationType
1	Kindle Fire HD	www.amazon.com	5	£99.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	£119.00	<input checked="" type="checkbox"/>	Pristine	ToBeBoughtProduct
2	XBOX 360 50G	www.ebay.com	4	£150.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	£135.00	<input checked="" type="checkbox"/>	Pristine	ToBeSoldProduct
3	ODROID-U3	www.robotics.com	5	£55.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	£55.00	<input checked="" type="checkbox"/>	Pristine	WishListProduct
20	WII U WIFI	www.Uk-deals.com	5	£200.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	£0.00	<input type="checkbox"/>	Pristine	ToBeBoughtProduct

Figure 5.29 The added product has been recommended successfully as it has a lower price

Table 5.1 summarises the testing stage by shown the class and functions tested as well as their outcomes:

Table 5.1 Summarised tests outcome

Class	Functionality tested	Outcome
UI class	The <i>Buy</i> tab and its functions	Success
	The <i>Sell</i> tab and its functions	Success
	The <i>Wishlist</i> tab and its functions	Success
	The <i>Recommendations</i> tab	Success
DB interactions	Execute a statement that returns a value	Success
	Execute a statement with nor return	Success
	Execute an update statement	Success
	The DB connection	Success
CommClass	Creates a file	Success
	Deletes a file	Success
	Modifies a file	Success
Message Exchange	The read modify and respond method	Success
Agents class	Calculate average	Success
The buying Agent	Deal with a file (sent/ received messages)	Success
	Claculate Average	Success
	Place recommendation	Success
	Create file	Success
The selling Agent	Uses the <i>MessageExchange</i> function	Success
AgentSys(main)	Instantiation of agents	Success
	Instantiation of UI	Success

## 5.2 Black box testing

The black box testing method is concerned with the overall behaviours and tasks of the system rather than testing how each and every single component works, therefore it is concerned with testing whether the high-level system performs that expected tasks or not.

The technique used to undertake this testing methodology: the system is expected to perform a number of tasks in a particular order from start up, the main tasks that are to be performed by the system are summarised into table 5.2 in a chronological order with the outcome of testing each task:

**Table 5.2 Black box test and outcomes**

<b>NO.</b>	<b>Excepted Task</b>	<b>Outcome</b>
1	The GUI is initialised and all tabs function as expected	Success
2	The buying agent is initialised	Success
3	The buying agent deleted files sent to the other agent once they have been read	Success
4	The agent checks if it received any messages and acts upon their contents	Success
5	The agent calculated the average value for each product	Success
6	The agent performed a recommendation based on the product details within the DB.	Success
7	The agent made sure that there is only one recommendation for each product within the DB	Success
8	The Agent has created a new file (as a message to the selling agent) each time a recommendation was made	Success

# Chapter 6 Verification

This stage is concerned with verifying whether the requirements defined in the requirements document are met or not, this is summarised in table 6.1, the outcome of this stage would be to verify whether the system solves the targeted problem by meeting all requirements or not.

**Table 6.1 verification of requirements**

Requirement	Met		Evidence	
	Design	Implementation	Design	Implementation
<b>Non-Functional Requirements</b>			<b>Page Numbers</b>	
Limited time for data collection from various sites	NO	NO	N/A	N/A
Limited notification and the option to disable them	YES	YES	45/47/57	62/71
Limiting the animations behaviour to a non-irritating level for the user	YES	NO	46	N/A
The agent must present info to the user in a user friendly manner	YES	YES	46/57	66
The use of hardware resource must be kept to minimum(memory, processor usage)	YES	NO	48	N/A
The failure rate must be within acceptable bounds	NO	NO	N/A	N/A
<b>Functional Requirements</b>				
Agents search for retail sites	YES	NO	53	N/A
Agents Extraction of data from sites	YES	NO	53	N/A
The Agents must be able to store information for later comparison	YES	YES	45/53/56	63/65
Agents rating of sellers	YES	NO	53	N/A
The Agents must be able to compare data from various sites and make recommendation accordingly	YES	YES	51/52/53	77
Present recommendations in a user-friendly approach	YES	YES	46/57	66
The Agents must be able to modify products within the DB	YES	YES	51/54/56	65
A wish list associated with the buying agent	YES	NO	52	N/A
Calculation of average of product values	YES	YES	51	76



Communication Requirements				
The development of a communication protocol	YES	YES	48	71/72
Implementation of communication methods	YES	YES	55	71/72
Message exchange between agents upon certain events	YES	YES	54	74
Security requirements				
Rate a site before recommending it	YES	NO	53	N/A
User Requirements				
A method of interacting with the system	YES	YES	45/46/47	63/65
A friendly user interface	YES	YES	46	65
Help options	YES	NO	46	N/A
Accessibility options	NO	NO	N/A	N/A
A number of configuration a user can set	YES	YES	47	70
User configuration required				
Disable notifications	YES	YES	45/46/47 49/51	63 – 70
Disable animations	YES	YES		
Limit the agents interaction to one site	YES	YES		
Limit the agents interaction to multiple sites	YES	YES		
User interface requirements				
Animated character representing each agent	YES	NO	46	N/A
Animated behaviour	NO	NO	N/A	N/A
Info outputted from agents must be presented to the user in a friendly way	YES	YES	46/57	66
A configuration menu to customize the systems settings	YES	YES	47	70
Other requirements				
A learning algorithm	NO	NO	N/A	N/A
A negotiation algorithm	NO	NO	N/A	N/A

As indicated from table 6.1 some requirements are not met during the development of the prototype, although that a design may have been already in place, these requirements will be met during the development of the finished product.

The requirements document provides details of the functionalities required to solve the tackled problem, as the majority of these requirements have been met then it is safe to say that the developed system does solve the problem it intended to solve, the testing stage confirms that the met requirements are tested and functioning as expected.

# Chapter 7 Validation

This stage is concerned with reviewing the functionalities implemented to assess whether those functionalities do truly solve the problem tackled through the requirements document and if such functionalities have been implemented correctly or need improvement

The validation stage is normally carried out by a validator who has not developed the system, however this is not the case here, as this is an independent project.

This section will only discuss the requirements that are met within the design or/and implementation stage, the requirements discussed are in table 7.1:

**Table 7.1 Validation of requirements**

Requirement	Review
<b>Non-Functional Requirements</b>	
Limited notification and the option to disable them	The user has been provided with the option to disable notification, the production of notification is kept to bear minimum. This is addressed throughout the design and implementation in stages such as DB and GUI design and implementation.
The agent must present info to the user in a user friendly manner	This requirement that was continuously considered throughout all stages of development, as the number of tasks the user a user has to undertake is kept to minimum and any method of user/system interaction was designed and implemented to be in the most user friendly way, such as GUI development.
The use of hardware resource must be kept to minimum(memory, processor usage)	The amount of resources the system uses could not be measured just yet, as not all required functionalities have been implemented yet
The failure rate must be within acceptable bounds	This may be assessed at the moment as there are still further functionalities to be implemented, however the functionalities currently implemented have been tested and are unlikely to fail in the future
<b>Functional Requirements</b>	
Agents search for retail sites	Plans and methods to achieve this has been included within the design stage but not in the implementation stage, the plans included seem to be sufficient to achieve this task at a high level however low level design plans are still required.
Agents Extraction of data from sites	
The Agents must be able to store information for later comparison	This has been met through the design and implementation of the database, comparison algorithms and the use of the JDBC API, however the implementation of comparisons algorithms are still required for the selling agent and the wish list
The Agents must be able to compare data from various sites and make recommendation accordingly	

Agents rating of sellers	A method of rating sites has been included within the design stage however it has not been implemented, the method also requires further elaboration at a lower level such as defining the libraries that can be used to achieve this.
Present recommendations in a user-friendly approach	Recommendation will always be displayed within a dedicated area on the GUI, rather than randomly appear in the form of popups, notifications are also presented through the task bar according to Microsoft suggestions therefore recommendations are presented in a user friendly manner.
The Agents must be able to modify products within the DB	This has been implemented, and there is no other better method than the current one used
A wish list associated with the buying agent	This has been considered in the design stage however it has not been implemented, although implementing such functionality is simplistic as the functionalities required by it are near identical to the current ones used by the buying agent
Calculation of average of product values	The current function in place, calculates the average price successfully however it has a limitation, the current function calculates the average price based on ten entries as an array of size ten is used, this limitation can be overcome by using vectors as their size does not have to be defined prior to runtime.
<b>Communication Requirements</b>	
The development of a communication protocol	A communication protocol has been developed and complied with throughout the development process, it is made according to various considerations(memory utilization and consistency )
Implementation of communication methods	This has been implemented, however it can be improved, the current method only allows for one file to be sent at a time by each agent as there is a high possibility that an agent may need to send multiple messages at a time.
Message exchange between agents upon certain events	The events which trigger message exchange have been defined however some communication methods such as negotiation are yet to be designed and implemented
<b>Security requirements</b>	
Rate a site before recommending it	Assuring that a site is of a suitable rating before recommending has already been implemented within the comparison and recommendation functionality
<b>User Requirements</b>	
A method of interacting with the system	This has led to the appearance of another requirement, which is a friendly user interface.
A friendly user interface	This has been considered throughout the design and implementation stages, however this requirement has not been fully achieved as a user guide and help options are yet to be included.
Help options	This has been considered during the design but has not been implemented and is highly required within the system as its aim to be a user friendly alternative to other e-commerce

	systems, without user assistance it will not be a friendly system as the user would have to fend for themselves
Accessibility options	This has not been considered during the design or implementation stages and needs to be considered during the development of the finished product
<b>User configuration required</b>	
Disable notifications	This has been met throughout the design and implementation of the GUI and the database, however the means to display notification and animations is not yet defined.
Disable animations	
Limit the agents interaction to one site	
Limit the agents interaction to multiple sites	
<b>User interface requirements</b>	
Animated behaviour	The animated behaviour to be performed by agents and its triggers have been discussed in the design, however it has not been discussed how these animation can be played
Info outputted from agents must be presented to the user in a friendly way	Both the user interface and notifications have been used to achieve this requirement as mentioned within the design, the user also has been provided with option to control information outputted by agents (disabling notification/animations)
A configuration menu to customize the systems settings	in order to allow the user to set the systems settings and have some control over its behaviour, a tab has been included within the GUI purely for this purpose, the tabs contents are synchronised with the database which contains a table that holds the systems settings, therefore agents can refer to this table in the database to behave accordingly to the setting placed by the user.
<b>Other requirements</b>	
A learning algorithm	These are MAS capabilities that if implemented will greatly enhance and perfect the implementation of the system.
A negotiation algorithm	
A User Guide	Yet to be written

This section can be reviewed prior to carrying out any future work as it identifies which functionalities are yet to be implemented and which can do with improvements.

# Chapter 8 Conclusion

The prototype has now been successfully implemented, containing the main functionalities of the system.

Successful implementation have been confirmed throughout the testing, verification and validation stages, which test all implemented functionalities and reviews them against the requirements of the system.

The system aims to provide an alternative to manually carrying out online purchases and sales, in a user friendly way suiting a wide range of users (computer illiterates, elderly users, new users), while demonstrating the use of a MASs and their beneficial capabilities, this meets the original aim set out prior to beginning the development process.

The objectives initially set out, have all been met as shown in table 8.1:

**Table 8.1 reviews the projects objectives**

Objective	Status
Automate e-commerce related tasks, such as searching for retail sites with a good rating that is based on user reviews	Partially Achieved
Efficiently use the properties and features of agent systems to achieve the systems aims.	Achieved
The functionalities of the system should be based upon tackling the negative aspects of the current manual process.	Achieved
The system must be as user friendly as possible in every aspect, as this is part of the systems aim.	Achieved
Assessing the feasibility of the project and risks.	Achieved
Analysing development tools, languages, methodologies and making appropriate choices	Achieved
Completion of the system by 04/04/2016	Achieved
Prototype development	Achieved
Verifying and validating whether the developed system contained the defined functionalities in order to assure the system has achieved its goal.	Achieved

Therefore it is safe to say that the system successfully tackles the problem it initially set to solve and meets both the aims and objectives initially set. Additional steps were taken to assure that the requirements identified are met and implemented in the most suitable method (the validation stage).

The system demonstrates some MASs capabilities and introduces methods of their use to automate e-commerce processes, the system shows how advantageous the use of agents can be. It is really unfortunate that the field is still within its infancy. The encouragement by specialists within this field to standardise agents and overcome obstacles which prevent its advancement is now understood and appreciated.

As mentioned in the introduction, the prototype contains the main functionalities of the system however not all designed or required functionalities have been implemented, this does not mean

that once the finished product is to be developed then everything has to be re-implemented, rather the development process can carry on from where the prototype left off.

The verification stage will assist in identifying which parts have already been implemented or designed and which parts are yet to be implemented or designed.

The system implemented could do with a more efficient implementation approach, some improvements have been suggested through the validation however the following improvements can be implemented in future work:

- The language used to implement the system was mainly JAVA which is object oriented, providing a wide range of features, the implementation of such features can be highly beneficial, for example Agents class contains functionality that is common to both agents however these functionalities may need slight modification when used by each agent, in such scenario inheritance and function overriding could have been used.
- The current targeted users are computer inexperienced, new users and elderly users as well as those who are tired of the current manual process, the set of targeted users does not have to remain static rather it can be expanded to include online based business that continuously carry out market research as such process can be potentially carried out by agents.
- The implementation of learning algorithm, which will cause the system to self-enhancement and a reduced failure rate, this means that the system developer or maintainer is not the only responsible
- The system currently does not identify sites to extract information from, this is one of the systems functionalities which are addressed in the design but are yet to be implemented in the future.
- The possible solution introduced in the early stages are potentially viable and it would be beneficial to embed them within the system during the future.

The functionalities implemented are not perfected, however the validation stage does review each implemented functionality and recommend improvements when needed, identified improvements can also be part of future work.

The use of a new methodology such as O-MaSE have proofed to have a highly beneficial impact on the development of the system, as progress was made through the methodologies stages and fragments it became easier to conceptualise the behaviour of the system and responsibilities of agents then produce a design accordingly.

The customised engineering approach used by O-MaSE have also proven to be highly beneficial as it provides engineers with the flexibility of modifying an existing methodology by only making use of stages that are relevant, as there are some stages encountered within the O-MaSE methodology that do not have any beneficial impact on the development process and unnecessarily consume time to be followed therefore they have simply been left out

It has been made clear in the literature review of how the current system differs from other MAS e-commerce systems that have been implemented, if it was not for the literature review

a system close or identical to already existing systems may have been implemented, if an identical system was to be released to the market then that may result in legal issues,

The system will be ready for market release once all the functionalities stated within the requirements document are met, additional improvements and proposed features can be added throughout the systems lifetime through updates and patches.

## Chapter 9 References

[[Placeholder1]]

Alonso, E., D'inverno, M., Kudenko, D., Luck, M. and Noble, J. (2001) 'Learning in multi-agent systems'. *The Knowledge Engineering Review*, vol. 16, no. 3, pp. 277 - 284.

Aylett, R., Brazier, F., Jennings, N., Luck, M., Nwana and Preist, C. (1998) 'Agent Systems AND Applications'. *The Knowledge Engineering Review*, vol. 13, no. 3, pp. 303 - 308.

Bădică, , Budimac, and Ivanović, (2011) *Software Agents: Languages, Tools, Platforms*. Software Engineering Department. Faculty of Automatics, Computers and.

Barbuceanu, M., Lo and W. (2000) 'A Multi-Attribute Utility Theoretic Negotiation Architecture for Electronic Commerce', International Conference on Autonomous AGENT.

Beer, M., D'inverno, M., Luck, M., Jennings, N., Preist, C. and Schorder, M. (1999) 'Negotiation In Multi-agent systems'. *The Knowledge Engineering Review*, vol. 14, no. 3, pp. 285 -290.

Bruno, E. (2011). Available at: <http://www.drdobbs.com/jvm/easy-dom-parsing-in-java/231002580>.

Collins, J., Ketter, W. and Gini, M. (2002) 'A Muti-agent Negotiation Tested for contractig Tasks'. *International Journal of Electronic Commerce*, vol. 7, no. 1, pp. 35-57.

Dastani, M., Jorge, J. and Gomez-Sanz (2005) 'Programming multi-agent systems'. *The knowledge Engineering Review*, vol. 20, no. 2, pp. 151 - 164.

Developers, J. (2015) *Jason*. Available at: <http://jason.sourceforge.net/wp/description/>.

Duma, D. (2014) *Agent-Based Systems*. Available at: <http://www.inf.ed.ac.uk/teaching/courses/abs/slides/intro-jason-2x2.pdf>.

Ferber, J., Gutknecht, O. and Fabien, M. (2004) 'From Agents to Organizations: An Organizational View'. *University of Montpellier II*, pp. 214-230.

FFA (2015) *FRAUD THE FACTS 2015*: Financial Fraud Actions.

fmichel (2016) *madkit*. Available at: <http://www.madkit.org/>.

Garcia-Ojeda, J.C., DeLoach, S.A. and Robby, W.H. (2007) *agentTool III Homepage*. Available at: <http://agenttool.cis.ksu.edu/>.

Helmy, T. (2007) 'Collaborative Multi-Agent-Based E-Commerce Framework'. *International Journal of Computersw, Systems and Signals*, vol. 8, no. 1.



homeandlearn 2015. Available at:  
[http://www.homeandlearn.co.uk/java/databases\\_and\\_java\\_forms.html](http://www.homeandlearn.co.uk/java/databases_and_java_forms.html) (Accessed: 2015).

Huns, M. and Singh, M. (1998) 'Readings in Agent Morgan Kaufmann Publishers'.

Kalliopi, K. and Nick, B. (2015) *A Survey of Agent Platforms*. Survey: Journal of Artificial Societies and Social Simulation 18.

Ltd, A.O.S.P. <http://www.agent-software.com.au/>. Available at: <http://www.agent-software.com.au/products/jack/> (Accessed: 2015).

Luck, M. (1997) 'Foundations of multi-agent systems: issues and directions', The Knowledge Engineering Review, pp. 307- 308.

Michael, W. (2012). Available at:  
<http://www.cs.ox.ac.uk/people/michael.wooldridge/teaching/robotics/agentspeak>.

Micheal, W. and David, K. (n.d) *The Gaia Methodology for Agent-Oriented Analysis and Design*. Netherlands: Kluwer Academic Publishers.

Minsky, M. (1986) *The society of mind*.

Mundhe, M. and Sen, s. (2000) 'Evaluating concurrent reinforcement learners', fourth international conference on multi agent systems, pp. 421 - 422.

Padgham, L. (n.d) *Introduction to Belief Desire Intention*. Available at:  
<http://goanna.cs.rmit.edu.au/~linpa/Presentations/BDIntro.pdf>.

Pierre-Michel , and Demazeau1, Y. (n.d) *From Analysis to Deployment*. Survey.

Pty., A.O.S. (1999) *JACK Intelligent Agents*: Agent Oriented Software Pty. Ltd.

Rao, A.S. (2000s) *AgentSpeak(L): BDI Agents speak out in a logical*. Australian Artificial Intelligence Institute.

Russell, S. and Norvig, P. (1995) 'Artificial Intelligence: A Modern Approach Prentice Hall'. *Coevolution and learning in multiagent systems AAAI press*.

StackOverflow (2015) *Developer Survey*. Available at:  
<http://stackoverflow.com/research/developer-survey-2016#technology-desktop-operating-system> (Accessed: 2016).

Stone, P. and Veloso, M. (1998) 'Towards collaborative and adversarial learning: A case study for robotic soccer'. *Internation journal of Human Computer Studies*, vol. 48, no. 1, pp. 83-104.

Sugwara, T. and Lesser, V. (1993) 'On-line learning of Coordinated Plans Computer Science Technical Report'. *Unoversity of Massachusettss*.



























Wei, Y., Moreau, L. and Jennings, N. (2003) 'Recommender System ', Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, pp. 600-607.




















Weiss , G. (1998) 'Multiagent system: A modern Approach to Distributed Artificial Intelligence'. *MIT Press*.

Zeng, (2009) 'An Agent-based online Shopping System in E-commerce'. *Canadian Centre of Science and Education*, vol. 2, no. 4, pp. 1 - 6.

# Chapter 10 Appendix

## Gantt chart

ID	 Task Mode	Task Name	Duration	Start	Finish
1		<b>Project Proposal</b>	<b>82.5 days</b>	<b>Mon 29/06/15</b>	<b>Wed 21/10/15</b>
2		Identify the main Problem and issues	2 days	Mon 29/06/15	Tue 30/06/15
3		Set out the aims and objective of the project as well as defining the deliverable upon completion of the project	2 days	Wed 01/07/15	Thu 02/07/15
4		Litrature Review	1 wk	Fri 03/07/15	Thu 09/07/15
5		Propose a number of solution	3 days	Fri 10/07/15	Tue 14/07/15
6		Comapre the propsed solutions and choose the most suitable one	2 days	Wed 15/07/15	Thu 16/07/15
7		Identify the requirments of the project	1 wk	Fri 17/07/15	Thu 23/07/15
8		Assess the feasibility of the chosen solution	2 wks	Fri 24/07/15	Thu 06/08/15
9		Identify the required knowledge and skill to undertake the project	3 days	Fri 07/08/15	Tue 11/08/15
10		Create A Preliminary Design	1 day	Wed 12/08/15	Wed 12/08/15
11		identify the resources required to implement the system	1 day	Thu 13/08/15	Thu 13/08/15
12		Meet with Project supervisor and review progress	1 day	Tue 06/10/15	Tue 06/10/15
13		FYP breifing Session	1 day	Mon 28/09/15	Tue 29/09/15
14	 	Produce a project definition report	75 days	Mon 29/06/15	Fri 09/10/15
15		Proof Read Intial Report	1 wk	Mon 12/10/15	Fri 16/10/15
16		Update Logbook	82.5 days	Mon 29/06/15	Wed 21/10/15
17					
18		<b>Progress Assesment</b>	<b>29 days</b>	<b>Fri 30/10/15</b>	<b>Wed 09/12/15</b>
19		Progress assesment briefing session	1 day	Fri 30/10/15	Fri 30/10/15
20		progress assesment presentation creation and preperation	1.2 wks	Mon 30/11/15	Sun 06/12/15
21		Prprogress assesment presentation	1 day	Wed 09/12/15	Wed 09/12/15
22					
23					
24		<b>Project Development</b>	<b>131 days</b>	<b>Mon 26/10/15</b>	<b>Mon 25/04/16</b>

ID	 Task Mode	Task Name	Duration	Start	Finish
25		Analysis of current development tools, methodologies and programming technology	2.6 wks	Mon 26/10/15	Wed 11/11/15
26		Choosing Development tools, tools and programming technologies based on analysis outcome	6 days	Tue 17/11/15	Tue 24/11/15
27		start of the design stage according to the methodology	6.2 wks	Sun 13/12/15	Fri 22/01/16
28		Design UI componenets	0.2 wks	Sun 14/02/16	Sun 14/02/16
29		Design Algorithms to implement functionalities	2 wks	Mon 15/02/16	Fri 26/02/16
30		Implement Functionalities according to designs	1.8 wks	Sun 28/02/16	Wed 09/03/16
31		Black box testing of functionalities	1.4 wks	Mon 07/03/16	Tue 15/03/16
32		White box testing of systems operation.	1 day	Wed 16/03/16	Wed 16/03/16
33		verification and validation of system	2 days	Thu 17/03/16	Fri 18/03/16
34		Write final report	84 days	Tue 01/12/15	Fri 25/03/16
35		Proof Read Intial Report	10 days	Mon 21/03/16	Fri 01/04/16
36		Update Logbook	84 days	Tue 01/12/15	Fri 25/03/16
37		Create User Guide	1 wk	Mon 28/03/16	Fri 01/04/16
38					
39		<b>Poster Presentation</b>	<b>30 days</b>	<b>Tue 15/03/16</b>	<b>Mon 25/04/16</b>
40		Poster breifing session	1 day	Tue 15/03/16	Tue 15/03/16
41		Project Poster creation	1.6 wks	Wed 06/04/16	Fri 15/04/16
42		Project Poster presentation	1 day	Mon 25/04/16	Mon 25/04/16

# System Code

## Uiclass.JAVA

```
public class Uiclass {

    public JFrame frmSystemUtilitie;
    private JTextField ToBeBoughtProductTF;
    private JTextField ToBeSoldProductTF;
    private JTextField AddRemoveWishlistProductTF;
    private JTextField OneSiteTF;
    private JTextField limitedSites1TF;
    private JTextField LimitedSites2TF;
    private JTextField LimitedSites3TF;
    private JTable RecommendationsTable;
    private JTable WishlistTable;
    private JTable table;

    public void AgentsGUI() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     * @wbp.parser.entryPoint
     */
    @SuppressWarnings("deprecation")
    private void initialize() {
        frmSystemUtilitie = new JFrame();
        frmSystemUtilitie.setTitle("System Utilitie");
        frmSystemUtilitie.setBounds(100, 100, 701, 445);
        frmSystemUtilitie.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmSystemUtilitie.getContentPane().setLayout(null);
        frmSystemUtilitie.setVisible(true);
        final String ucanaccessString = "jdbc:ucanaccess://h:/FYP/Implementation Files/Project Files/AgentsDataBase/AgentsDataBase.accdb";
        JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);
        tabbedPane.setBounds(10, 11, 665, 385);
        frmSystemUtilitie.getContentPane().add(tabbedPane);

        ///////////////////////////////////////////////////
        //Buy Tab//

        JPanel BuyingTab = new JPanel(); // Declares a new tab
        tabbedPane.addTab("Buy", null, BuyingTab, null);
        BuyingTab.setLayout(null); // Adds the new tab to the window

        ToBeBoughtProductTF = new JTextField(); // a text field to hold the new products value
        ToBeBoughtProductTF.setBounds(32, 58, 227, 20);
        BuyingTab.add(ToBeBoughtProductTF);
        ToBeBoughtProductTF.setColumns(10);

        JButton AddToBeBoughtProductB = new JButton("Submit"); // declaring the button
        AddToBeBoughtProductB.setBounds(296, 57, 89, 23);
        BuyingTab.add(AddToBeBoughtProductB);

        AddToBeBoughtProductB.addActionListener(new ActionListener() { // executes upon button press
            public void actionPerformed(ActionEvent arg0) {

                String stmt = "INSERT INTO ToBeBoughtProducts (ProductName, ProductStatus) VALUES('"+ToBeBoughtProductTF.getText()+"','"+True')";
                DBinteractions AddToBeBoughtProducts = new DBinteractions();
                AddToBeBoughtProducts.ExecuteDBQueryNoReturn(stmt);
            }
        });

        JLabel lblNewLabel = new JLabel("Enter the description or name of the product you wish to buy in the text field below:");
        lblNewLabel.setBounds(10, 24, 411, 23);
        BuyingTab.add(lblNewLabel);
    }
}
```

```

////////////////////////////////////
////Selling Tab/////

JPanel SellingTab = new JPanel();
tabbedPane.addTab("Sell", null, SellingTab, null);
SellingTab.setLayout(null);

Label label = new Label("Enter the description or name of the product you wish to sell in the text field below:");
label.setBounds(10, 24, 528, 23);
SellingTab.add(label);

ToBeSoldProductTF = new JTextField();
ToBeSoldProductTF.setBounds(32, 58, 227, 20);
SellingTab.add(ToBeSoldProductTF);
ToBeSoldProductTF.setColumns(10);

JButton AddToBeSoldProductB = new JButton("Submit");
AddToBeSoldProductB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        String stmtnt = "INSERT INTO ToBeSoldProducts (ProductName, ProductStatus) VALUES('"+ToBeSoldProductTF.getText()+"', 'True')";
        DBinteractions AddToBeSoldProducts = new DBinteractions();
        AddToBeSoldProducts.ExecuteDBQueryNoReturn(stmtnt);
    }
});
AddToBeSoldProductB.setBounds(296, 58, 89, 23);
SellingTab.add(AddToBeSoldProductB);

////////////////////////////////////
////Wishlist Tab/////

JPanel WishlistTab = new JPanel(); // declares a new tab
tabbedPane.addTab("Wishlist", null, WishlistTab, null);
WishlistTab.setLayout(null);

Label label_1 = new Label("To Add/Remove an item from/to the wishlist enter the items name in the box below:");
label_1.setBounds(22, 226, 518, 22);
WishlistTab.add(label_1);

AddRemoveWishlistProductTF = new JTextField(); // a text field to hold the product name (string)
AddRemoveWishlistProductTF.setBounds(55, 254, 195, 20);
WishlistTab.add(AddRemoveWishlistProductTF);
AddRemoveWishlistProductTF.setColumns(10);

JPanel panel = new JPanel();
panel.setBounds(22, 49, 611, 151);
WishlistTab.add(panel);
panel.setLayout(null);

JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(0, 0, 611, 151);
panel.add(scrollPane);

WishlistTable = new JTable(); // a table to display current wishlist items
scrollPane.setViewportView(WishlistTable); //encapsulates the table within a scroll pane to enable scrolling!

String stmtent = "SELECT ProductName FROM WishListProducts"; //selects data from the wishlist table
DBinteractions PopulateWishlistTable = new DBinteractions(); //
WishlistTable.setModel(DbUtils.resultSetToTableModel(PopulateWishlistTable.ExecuteDBQuery(stmtent))); // uses the dbutils method to populate

JButton SubmitWishlistB = new JButton("Submit"); // declares a new button
SubmitWishlistB.setBounds(301, 253, 89, 23);
WishlistTab.add(SubmitWishlistB);

```

```

SubmitWishlistB.addActionListener(new ActionListener() { // executes upon button press
    public void actionPerformed(ActionEvent arg0) {
        try {

            String stmtnt = "SELECT * FROM WishListProducts WHERE ProductStatus = Yes"; //retrieve all data

            DBinteractions AddRemoveWishlistProducts = new DBinteractions();
            //ResultSet rs = AddRemoveWishlistProducts.ExecuteDBQueryMR(stmtnt);
            ResultSet rs = AddRemoveWishlistProducts.ExecuteDBQuery(stmtnt);
            String ValueInBox = AddRemoveWishlistProductTF.getText();

            while (rs.next()){ // reads every row

                String ProductName = rs.getString("ProductName"); // stores retrieved data for later processing from the "ProductName" column
                System.out.println(ProductName);

                if (ProductName.equals(ValueInBox) ){ // if it exists within the DB then delete it

                    stmtnt = "DELETE FROM WishListProducts WHERE ProductName ="+" "+ValueInBox+" "+"";
                    AddRemoveWishlistProducts.ExecuteDBQueryUpdate(stmtnt);
                }
                else if (ProductName != ValueInBox){ // if it does not exist within the DB then add it

                    stmtnt = "INSERT INTO WishListProducts (ProductName, ProductStatus) VALUES ("+" "+ValueInBox+" "+", 'True')";
                    AddRemoveWishlistProducts.ExecuteDBQueryNoReturn(stmtnt);
                }
            }
        } catch (SQLException e) {

            e.printStackTrace();
        }
    }
});

JLabel lblNewLabel_1 = new JLabel("The table below shows current wishlist products:");
lblNewLabel_1.setBounds(22, 31, 422, 14);
WishlistTab.add(lblNewLabel_1);

////////////////////////////////////
///Recommendations Tab///

JPanel RecommendationsTab = new JPanel();
tabbedPane.addTab("Recommendations", null, RecommendationsTab, null);
RecommendationsTab.setLayout(null);

final JRadioButton ToBeBoughtRB = new JRadioButton("To Be Bought");
ToBeBoughtRB.setBounds(90, 77, 109, 23);
RecommendationsTab.add(ToBeBoughtRB);

final JRadioButton ToBeSoldRB = new JRadioButton("To Be Sold");
ToBeSoldRB.setBounds(277, 77, 109, 23);
RecommendationsTab.add(ToBeSoldRB);

final JRadioButton WishlistProductsRB = new JRadioButton("Wishlist Products");
WishlistProductsRB.setBounds(459, 77, 109, 23);
RecommendationsTab.add(WishlistProductsRB);

JLabel lblNewLabel_2 = new JLabel("Below is the holding any recommendations made by the system:");
lblNewLabel_2.setBounds(10, 20, 418, 14);
RecommendationsTab.add(lblNewLabel_2);

JTextArea lblNewLabel_3 = new JTextArea("Info displayed within the table below can be formatted by selecting by enabling the buttons below:");
lblNewLabel_3.setFont(new Font("Tahoma", Font.PLAIN, 11));
lblNewLabel_3.setLineWrap(true);
lblNewLabel_3.setEditable(false);
lblNewLabel_3.setBackground(UIManager.getColor("Button.background"));
lblNewLabel_3.setBounds(20, 45, 587, 23);
RecommendationsTab.add(lblNewLabel_3);

Panel panel_1 = new Panel();
panel_1.setBounds(22, 108, 603, 228);
RecommendationsTab.add(panel_1);
panel_1.setLayout(null);

JScrollPane scrollPane_1 = new JScrollPane();
scrollPane_1.setBounds(0, 0, 603, 228);
panel_1.add(scrollPane_1);

```

```

table = new.JTable();
scrollPane_1.setViewportViewView(table);

String stmtnt = "SELECT ProductName, Source, SourceRating, ProductValue, AverageValue, StockAvailability, RecommendationType, "
+ "ProductCondition FROM RecommendedProducts";
DBinteractions PopulateTheRecommendedProductsTable = new DBinteractions();
table.setModel(DbUtils.resultSetToTableModel(PopulateTheRecommendedProductsTable.ExecutedDBQuery(stmtnt))); // populates the recommendations table

WishlistProductsRB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        ToBeBoughtRB.setSelected(false);
        ToBeSoldRB.setSelected(false);

        String stmtnt = "SELECT ProductName, Source, SourceRating, ProductValue, AverageValue, StockAvailability, RecommendationType, ProductCor
DBinteractions PopulateTheRecommendedProductsTableFiltered = new DBinteractions();
        table.setModel(DbUtils.resultSetToTableModel(PopulateTheRecommendedProductsTableFiltered.ExecutedDBQuery(stmtnt))); // populates the recc

    }
});

ToBeBoughtRB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        WishlistProductsRB.setSelected(false);
        ToBeSoldRB.setSelected(false);

        String stmtnt = "SELECT ProductName, Source, SourceRating, ProductValue, AverageValue, "
+ " StockAvailability, RecommendationType, ProductCondition FROM RecommendedProducts"
+ " WHERE RecommendationType = 'ToBeBoughtProduct' ";
        DBinteractions PopulateTheRecommendedProductsTableFiltered = new DBinteractions();
        table.setModel(DbUtils.resultSetToTableModel(PopulateTheRecommendedProductsTableFiltered.ExecutedDBQuery(stmtnt))); // populates the recomm

    }
});

////////////////////////////////////
/// Settings Tab///

JPanel SettingsTab = new JPanel();
tabbedPane.addTab("Settings", null, SettingsTab, null);
SettingsTab.setLayout(null);

final JCheckBox SingleInteractionCB = new JCheckBox("Limit Interactions to a single site");
final JCheckBox MultipleInteractionCB = new JCheckBox("Limit interactions to multiple sites");
final JCheckBox GlobalInteractionCB = new JCheckBox("Enable global site interactions");

OneSiteTF = new JTextField();
OneSiteTF.setBounds(46, 114, 329, 20);
SettingsTab.add(OneSiteTF);
OneSiteTF.setColumns(10);
OneSiteTF.setEditable(false);

SingleInteractionCB.addActionListener(new ActionListener() {
    @SuppressWarnings("deprecation")
    public void actionPerformed(ActionEvent e) {

        if (SingleInteractionCB.isSelected()){
            MultipleInteractionCB.setSelected(false);
            GlobalInteractionCB.setSelected(false);
            OneSiteTF.setEditable(true);
        }
        else{
            OneSiteTF.setEditable(false);
        }
    }
});

limitedSites1TF = new JTextField();
limitedSites1TF.setBounds(46, 171, 329, 20);
SettingsTab.add(limitedSites1TF);
limitedSites1TF.setColumns(10);
limitedSites1TF.setEditable(false);

limitedSites2TF = new JTextField();
limitedSites2TF.setBounds(47, 202, 328, 20);
SettingsTab.add(limitedSites2TF);
limitedSites2TF.setColumns(10);
limitedSites2TF.setEditable(false);

limitedSites3TF = new JTextField();
limitedSites3TF.setBounds(47, 233, 328, 20);
SettingsTab.add(limitedSites3TF);
limitedSites3TF.setColumns(10);
limitedSites3TF.setEditable(false);

SingleInteractionCB.setBounds(22, 84, 239, 23);
SettingsTab.add(SingleInteractionCB);

```



```

MultipleInteractionCB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        if(MultipleInteractionCB.isSelected()){
            SingleInteractionCB.setSelected(false);
            GlobalInteractionCB.setSelected(false);
            LimitedSites3TF.setEditable(true);
            LimitedSites2TF.setEditable(true);
            LimitedSites1TF.setEditable(true);
        }
        else{
            LimitedSites3TF.setEditable(false);
            LimitedSites2TF.setEditable(false);
            LimitedSites1TF.setEditable(false);
        }
    }
});
MultipleInteractionCB.setBounds(22, 141, 376, 23);
SettingsTab.add(MultipleInteractionCB);

JButton ApplySettingsB = new JButton("Apply");
ApplySettingsB.setBounds(492, 263, 141, 23);
SettingsTab.add(ApplySettingsB);

JButton CancelSettingsB = new JButton("Cancel");
CancelSettingsB.setBounds(492, 304, 141, 23);
SettingsTab.add(CancelSettingsB);

JButton DefaultsB = new JButton("Revert to defaults");
DefaultsB.setBounds(492, 220, 141, 23);
SettingsTab.add(DefaultsB);

JCheckBox AnimationsCB = new JCheckBox("Disable animations");
AnimationsCB.setBounds(435, 36, 134, 23);
SettingsTab.add(AnimationsCB);

JCheckBox NotificationsCB = new JCheckBox("Disable notification");
NotificationsCB.setBounds(435, 84, 157, 23);
SettingsTab.add(NotificationsCB);
}
}

```

## DBInteractions.JAVA

```

public class DBInteractions {

    /*
     * 1. A function that executes a SQL Statement returns a value
     * 2. A Function that executes a SQL Statement and loops with a return value
     * 3. A function that executes a SQL Statement an update statement required for delete operations
     *
     * NOTE: connection and statement object are deliberately not included as private class variables as they
     *       are meant to be local for each function
     */
    final private String ucannaccessString = "jdbc:ucanaccess://h:/FYP/Implementation Files/Project Files/AgentsDataBase/AgentsDataBase.accdb";

    // executes a SQL statement and returns the result of the execution
    public ResultSet ExecuteDBQuery(String SQLst){

        try{

            Connection conn =DriverManager.getConnection(ucannaccessString); // DB connection
            Statement s = conn.createStatement(); // declaring a statement object (from JDBC lib)
            ResultSet rs = s.executeQuery(SQLst);

            return rs;

        }catch(Exception e){
            e.printStackTrace();
            return null;
        }

    }

}

```

```

public ResultSet ExecuteDBQueryMR(String SQLst){ // redundant and does not work it will work if these values are loaded into an array as your program
    try{
        Connection conn = DriverManager.getConnection(ucannaccessString); // DB connection
        Statement s = conn.createStatement(); // declaring a statement object (from JDBC lib)
        ResultSet rs = s.executeQuery(SQLst);
        rs.isBeforeFirst();
        while(rs.next()){return rs;}
    }catch(Exception e){
        e.printStackTrace();
        return null;
    }
}

// Executes a statement without returning a result
public void ExecuteDBQueryNoReturn(String SQLst){
    try{
        Connection conn = DriverManager.getConnection(ucannaccessString); // DB connection
        Statement s = conn.createStatement(); // declaring a statement object (from JDBC lib)
        s.executeUpdate(SQLst); // execute a statement without returning anything
        s.close();
    }catch(Exception e){
        e.printStackTrace();
    }
}

// can be used to pass statements that contains Delete or Update Functionality
public void ExecuteDBQueryUpdate(String SQLst){
    try{
        Connection conn = DriverManager.getConnection(ucannaccessString); // DB connection
        Statement s = conn.createStatement(); // declaring a statement object (from JDBC lib)
        s.executeUpdate(SQLst);
        s.close();
    }catch(Exception e){
        e.printStackTrace();
    }
}
}

```

## CommClass.JAVA

```

public class CommClass {
    /*
     * 1. A function to delete files when read
     * 2. A function to create files (protocol compliant)
     * 3. A function that modifies the file contents to change message status
     */

    public void DeleteFileWhenRead(File f) throws ParserConfigurationException // This Function should be called by the agent that wants to delete a file
    , SAXException, IOException
    {
        String MessageStatus = "";
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

        DocumentBuilder builder = factory.newDocumentBuilder();
        org.w3c.dom.Document doc = builder.parse(f); // remember this should be the file path

        NodeList list = doc.getElementsByTagName("CommStatus");

        System.out.println(list.getLength());

        for (int i = 0; i < list.getLength(); i++) {
            Element item = (Element)list.item(i);
            MessageStatus = (String) item.getFirstChild().getNodeValue();
        }

        if(MessageStatus.equals("Read"))
        {
            f.delete();
        }
    }
}

```

```

// requires the file name of the file that should be modified passed to it
public void MarkMessageAsRead(File f) throws IOException, SAXException, ParserConfigurationException{
    if(f.exists() && !f.isDirectory()) {
        //System.out.println("the file exists");

        String MessageStatus="";
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse(f); // "C:/Users/Aboudi's/Desktop/ToSellingAgent.xml" //not needed use f instead
        NodeList list = doc.getElementsByTagName("CommStatus");

        //System.out.println(list.getLength());

        for (int i = 0; i < list.getLength(); i++) {

            Element item = (Element)list.item(i);
            //System.out.print(((Node) item).getFirstChild().getNodeValue());
            MessageStatus = (((Node) item).getFirstChild().getNodeValue());
            item.getFirstChild().setTextContent("Read");

            OutputFormat outFormat = new OutputFormat(doc);
            FileOutputStream outStream = new FileOutputStream(f);
            XMLSerializer Serializer = new XMLSerializer(outStream,outFormat);
            Serializer.serialize(doc);

        }
    }
    else {
        System.out.println("does not exist exists");
    }
}

}

//To send in the name of the product to be recommended and the stated receptionest (ToSellingAgent)
public void CreateFile (String ProductName, String Receptionest) throws ParserConfigurationException, IOException{

    try {
        DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
        Document xmlDoc = docBuilder.newDocument();

        Text ProductNameToF = xmlDoc.createTextNode(ProductName);
        Text CommType = xmlDoc.createTextNode("MessageExchange");
        Text ReadStatus = xmlDoc.createTextNode("Unread");

        Element rootElement = xmlDoc.createElement("CommType");
        rootElement.appendChild(CommType);

        Element Status = xmlDoc.createElement("CommStatus");
        Status.appendChild(ReadStatus);

        Element Value = xmlDoc.createElement("CommValue");
        Value.appendChild(ProductNameToF); // becomes the child of the tag Value

        rootElement.appendChild(Status); // becomes the comm type of the tag Value
        rootElement.appendChild(Value); // becomes the commtype of the tag Value

        xmlDoc.appendChild(rootElement); // adds the root elements to the document
        OutputFormat outFormat = new OutputFormat(xmlDoc);

        File xmlFile = new File("C:/Users/Aboudi's/Desktop/"+Receptionest+".xml");
        FileOutputStream outStream = new FileOutputStream(xmlFile);
        XMLSerializer Serializer = new XMLSerializer(outStream,outFormat);
        Serializer.serialize(xmlDoc);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}

```

## MessageExchange.JAVA

```

public class MessageExchange {

    /*
     * This class should inherit from the CommsClass
     * 1. A function that reads and responds to file contents
     *   - Checks if it should delete it
     *   - Checks if it exists
     *   - Gets the tag contents
     *   - Acts upon contents
     */
    //File f = new File("C:/Users/Aboudi's/Desktop/ToSellingAgent.xml");

    //parameters(name and directory of the file you want the agent to read,the name of this agent)
    public void MonitorReadRespondModify(File f, String AgentName) throws ParserConfigurationException, SAXException, IOException{

        if(!f.getName().contains(AgentName)){ // if it does not contain the name then it must be the sender, incharge of deleting when read
            // then check if the file is read, if its is delete if it is not then do nothing
            CommsClass CheckIfItIsRead = new CommsClass();
            CheckIfItIsRead.DeleteFileWhenRead(f);
        }
        else if(f.exists() && !f.isDirectory()) {

            String ProductNameInFile=""; // to hold the value stored within the value tag
            String CommStatusInFile=""; // to hold the value stored within the CommStatus tag

            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.parse(f); // "C:/Users/Aboudi's/Desktop/ToSellingAgent.xml" //not needed use f instead

            NodeList list = doc.getElementsByTagName("CommValue");
            NodeList list2 = doc.getElementsByTagName("CommStatus");

```

```

for (int i = 0; i < list.getLength(); i++) {
    Element item = (Element)list.item(i);
    Element item2 = (Element)list2.item(i);

    //System.out.print(((Node) item).getFirstChild().getNodeValue());
    ProductNameInFile = (((Node) item).getFirstChild().getNodeValue());
    CommStatusInFile = (((Node) item2).getFirstChild().getNodeValue());

    String SalesTable = "ToBeSoldProducts";
    String BuyingTable = "ToBeBoughtProducts";
    String VariableNeutral = "";

    if(f.getName().contains("Buying")){
        VariableNeutral = SalesTable;
    }else if(f.getName().contains("Buying")){
        VariableNeutral = BuyingTable;
    }

    if(CommStatusInFile.equals("Read")){ // if the tag within the file is read then don't add it
        break;
    }
    else{ /// if the tag within the file is not read then add it
        //String stmtnt40 = "INSERT INTO "+VariableNeutral+" "+" "+" (ProductName, ProductStatus) VALUES ('"+ProductNameInFile+"', 'True')";
        String stmtnt40 = "INSERT INTO ToBeBoughtProducts (ProductName, ProductStatus) VALUES ('"+ProductNameInFile+"', 'True')"; // this li
        DBinteractions UpdateFileContentstoDB = new DBinteractions();
        UpdateFileContentstoDB.ExecutedBQueryNoReturn(stmtnt40);
        // then change the tag value by calling the modify

        CommClass ModifyIt = new CommClass();
        ModifyIt.MarkMessageAsRead(f);
    }
}
}
else {
    System.out.println("does not exist exists");
}
}

```

## Agents.JAVA

```

public class Agents {

    /*
     * 1. calculates the average of a product value of a set of products
     */

    //The agent which the average is going to be calculated for
    public void CalculateAverageValue() throws SQLException{ // calculates the average for each product in the DB

        DBinteractions Execute = new DBinteractions(); // a new instance to allow for use of methods

        String stmtnt = "SELECT ProductName FROM ToBeBoughtProducts WHERE ProductStatus = 'True'";
        ResultSet rs = Execute.ExecutedBQuery(stmtnt); // returns and store the results

        float[] Value = new float[10]; // to store the
        float AverageValue=0; // to hold the average value
        float cumulative = 0; // stores the cumulative of value
        int i=0; // incrementer to keep track of values

        while(rs.next()){
            String ProductName = rs.getString("ProductName");
            //System.out.print(ProductName);// it works so far

            String stmtnt2 ="SELECT ProductValue From ToBeBoughtProductDetails WHERE ProductName = '"+ProductName+"';//"+ProductName+"WII U WIFI
            ResultSet rs2 = Execute.ExecutedBQuery(stmtnt2);

            while(rs2.next()){
                float ProductValue = rs2.getFloat("ProductValue");
                //System.out.println("ProductValue var = " +ProductValue);
                Value[i] = ProductValue;
                cumulative = cumulative+ Value[i];
                ++i;
                AverageValue= cumulative/i;

                //System.out.println(i);
            }

            System.out.println(" AverageValue of " + ProductName+ " is " +AverageValue);
        }
    }
}

```

# TheBuyingAgent.JAVA

```
public class TheBuyingAgent {

    private CommClass CreateFileUponRec = new CommClass();
    //private String ProductName = "";
    private MessageExchange HandleMessages = new MessageExchange();
    private static File f = new File("C:/Users/Aboudi's/Desktop/ToBuyingAgent.xml");

    /*
     * 1. calls the calculate average function
     * 2. Performs a recommendation and creates a new file
     * 3. calls the function to deal with existing messages
     * 4. All called within one method
     */

    public void InitialiseAndLaunch(){

        try {

            // this should calculate the average for all products

            // this should loop independantly from any other statement
            HandleMessages.MonitorReadRespondModify(f, "BuyingAgent");
            PerfromRecommendation();

        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    private void PerfromRecommendation() throws ParserConfigurationException, IOException{

        DBInteractions Execute = new DBInteractions();
        try{
            String stmt1 = "SELECT ProductName FROM ToBeBoughtProducts WHERE ProductStatus = 'True'";
            //ResultSet rs = s.executeQuery(stmt1); // execute a statement without returning anything
            ResultSet rs= Execute.ExecuteDBQuery(stmt1);

            while(rs.next()) {
                String ProductName = rs.getString("ProductName");

                String stmt2 ="SELECT * From ToBeBoughtProductDetails WHERE ProductName = '"+ProductName+"'"; //"+ProductName+"WII U WIFI
                ResultSet rs3= Execute.ExecuteDBQuery(stmt2);

                String Source;
                float SourceRating;
                float ProductValue;
                boolean DeliveryAvailability;
                boolean StockAvailability;

                String ProductNameInRT;
                float SourceRatingInRT;
                float ProductValueInRT;
                boolean DeliveryAvailabilityInRT;
                boolean StockAvailabilityInRT;

                while(rs3.next()){
                    // these are the values from the ProductDetailsTable
                    Source= rs3.getString("Source");
                    SourceRating= rs3.getFloat("SourceRating");
                    ProductValue= rs3.getFloat("ProductValue");
                    DeliveryAvailability= rs3.getBoolean("DeliveryAvailability");
                    StockAvailability= rs3.getBoolean("StockAvailability");

                    //if((ProductValue <= AverageValue )&&( SourceRating>= 2.5)&&(DeliveryAvailability=true)&&(StockAvailability=true)){ // defining the recom
                    if(( SourceRating>= 2.5)&&(DeliveryAvailability=true)&&(StockAvailability=true)){ // defining the recommendation criteria
                        String stmt3 ="SELECT * From RecommendedProducts WHERE ProductName = '"+ProductName+"' AND RecommendationType='ToBeBoughtProduct'";
                        ResultSet rs4 = Execute.ExecuteDBQuery(stmt3);

                        /*if(rs4.isNull()){ // this is to be used to check whether the product exists within the DB or not, if not it adds it
                            String stmt5 = "INSERT INTO RecommendedProducts (ProductName, Source, SourceRating,ProductValue,DeliveryAvailability,StockAvailabil
                            System.out.println("this should work");
                            s5.execute(stmt5);
                            // call the create file method
                        }*/ // ask raddi

                        while(rs4.next()){

                            ProductNameInRT= rs4.getString("ProductName");
                            SourceRatingInRT= rs4.getFloat("SourceRating");
                            ProductValueInRT= rs4.getFloat("ProductValue");
                            DeliveryAvailabilityInRT= rs4.getBoolean("DeliveryAvailability");
                            StockAvailabilityInRT= rs4.getBoolean("StockAvailability");

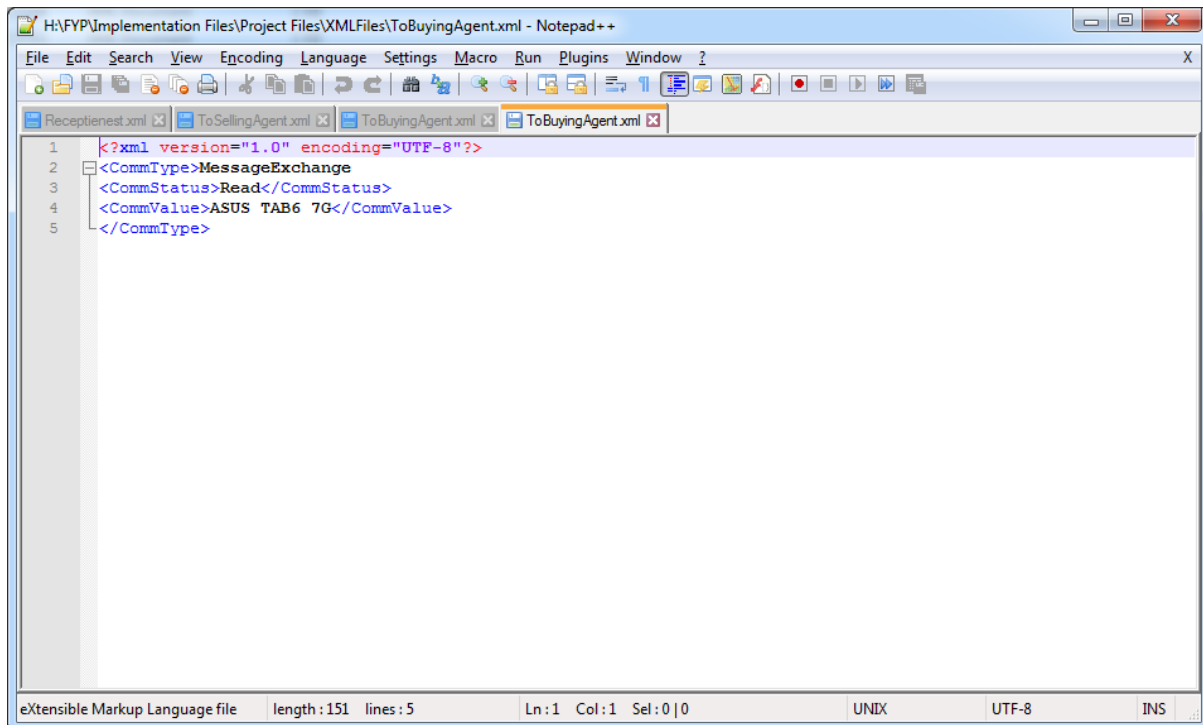
                            System.out.println("penguin");
                            System.out.println(ProductNameInRT);
                            System.out.println(ProductValue);
                            System.out.println(Source);
                            System.out.println(SourceRating);
                            System.out.println(DeliveryAvailability);
                            System.out.println(StockAvailability); //keep for trouble shooting purpouses then delete it
                        }
                    }
                }
            }
        }
    }
}
```

## TheSellingAgent.JAVA

## AgentSys(main).JAVA

118

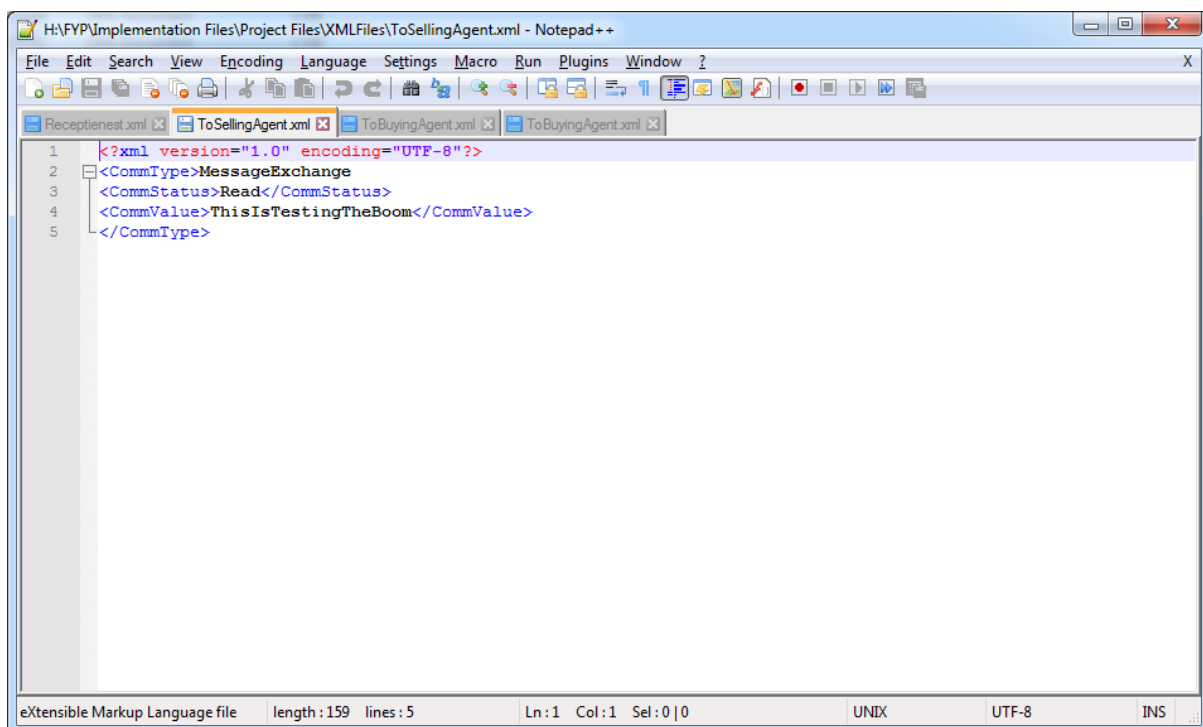
## XML communication files



The screenshot shows a Notepad++ window titled "H:\FYP\Implementation Files\Project Files\XMLFiles\ToBuyingAgent.xml - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and Help. The toolbar contains various icons for file operations and editing. The tab bar shows four open files: Receptionist.xml, ToSellingAgent.xml, ToBuyingAgent.xml, and ToBuyingAgent.xml. The main text area displays the following XML code:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <CommType>MessageExchange
3 <CommStatus>Read</CommStatus>
4 <CommValue>ASUS TAB6 7G</CommValue>
5 </CommType>
```

The status bar at the bottom indicates: eXtensible Markup Language file, length: 151, lines: 5, Ln: 1, Col: 1, Sel: 0 | 0, UNIX, UTF-8, INS.



The screenshot shows a Notepad++ window titled "H:\FYP\Implementation Files\Project Files\XMLFiles\ToSellingAgent.xml - Notepad++". The menu bar and toolbar are identical to the previous screenshot. The tab bar shows the same four files. The main text area displays the following XML code:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <CommType>MessageExchange
3 <CommStatus>Read</CommStatus>
4 <CommValue>ThisIsTestingTheBoom</CommValue>
5 </CommType>
```

The status bar at the bottom indicates: eXtensible Markup Language file, length: 159, lines: 5, Ln: 1, Col: 1, Sel: 0 | 0, UNIX, UTF-8, INS.