# MULTI AGENT SYSTEM FOR EVOLVING BUSINESS ENVIRONMENTS

P.L.A.UDAYANGA RATHNASEKARA

168292M

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa

Sri Lanka

March 2019

# MULTI AGENT SYSTEM FOR EVOLVING BUSINESS ENVIRONMENTS

P.L.A.UDAYANGA RATHNASEKARA

168292M

Thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa

Sri Lanka

March 2019

## Declaration

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name of the student: Signature:

P.L.A Udayanga Rathnasekara.

Date:

The above candidate has carried out research for the Master's thesis under my supervision.

Name of the supervisor: Signature:

Prof. A.S. Karunananda.

Date:

## Acknowledgements

I would like to express my since gratitude to my supervisor Prof. A.S. Karunananda for providing me invaluable guidance, comments and feedback throughout the entire project. I would also like to acknowledge him for constantly motivating me to work harder to make this study a success.

My sincere thanks goes to all the lecturers of the Department of Computational Mathematics for their insightful comments and encouragement. Without their precious support it would not be possible to conduct this research.

Furthermore, I would like to thanks my wife, parents and family members, fellow colleagues and friends for the support they have given me for successfully complete this research project.

## Table of Contents

**List of Tables**

**List of Figures**

## List of Abbreviations

| Abbreviations | Meaning |
|---|---|
| API | Application Programming Interfaces |
| ECA | Electronic Commerce Association |
| GA | Genetic Algorithms |
| GB | Gigabyte |
| GUI | Graphical User Interface |
| JADE | Java Agent DEvelopment Framework |
| JDK | Java Development Kit |
| JPA | Java Persistence API |
| JSP | Java Server Page |
| MAS | Multi Agent System |
| MASCB | Multi Agent System for Customer Bargaining |
| RAM | Random Access Memory |
| SCM | Supply Chain Management |
| SET | Secure Electronic Transaction |
| SSL | Socket Layer Protocol |

# Abstract

There are numerous online merchants and digital payment systems with loyalty offers for customers. However, limited research has been conducted to introduce customer driven bargaining based on profile of the customer. This matter is of great importance since some customers are more important than others and cannot be treated equally with common rewards. We have conducted a research to implement Multi Agent Based merchant-customer bargaining solution for online transactions.

This solution comprises of four types of Agents, namely, Customer Agents, Manager Agent, Search Agent and Merchant Agents. Upon the request by a customer agent with the product name, customer location, history of purchase pattern of the customer, multiple merchant agents will be activated. A genetic algorithm process is used for selecting the most suitable merchant in the echo system by considering the distance and previous customer reviews for the merchants. Consequently, agents start communication on the product and start negotiations to relax the constraints and finally agreed upon a bargain price acceptable to both customer and the successful merchant. The solution has been implemented on JADE. It has been tested since developing the system and found some encouraging results, which has generated experience for a formal evaluation.

Even though there are lots of merchant controlling bargaining systems in the market this system introduces a customer driven bargaining system. This customer driven bargaining system may make people life easier and may bring more advantages for both customers and merchants in an echo system. MAS based solution is ideal for this kind of complex system and it handles the complexity of negotiation, coordination and communication of the dynamic environment. Platform may provide ideal solutions and recommendations based on the past data system keeps and the APIs of developed system, can be used to connect with any existing mobile and e-commerce App to bring this feature as a value added service to their customers. This can inevitably lead to greater levels of customer satisfaction and help the merchants achieve a competitive edge over other competitors.

Chapter 1

# Introduction

## 1.1 Prolegomena

E-Shopping has become a fad during the last few decades owing to its great flexibility of buying from home or office. The modern business environment is dynamic and constantly changing, however customers are expecting their specific requirements to be met at satisfactory level [1]. Software agent technologies have taken the new-generation e- commerce to a different level, saving more time of customers who would have otherwise taken the hassle of making visits to physical stores [2][3]. Agent based e-commerce has come into being and is now the focus of the next generation e-commerce [3].

Bargaining is meant to negotiate a price between two parties. People use bargaining in any kind of market ranging from a rural level boutique to the international market. In the present context, there is a widely shared enthusiasm over online shopping, so that numerous digital payment platforms are available to online shoppers. Even though there are many digital payment methods and online merchants, it is an arduous task to look for a merchant who facilitates bargaining except the traditional auction supporting merchants.

AI applications like Alexa, Siri, and customer service chatbots has gain the attention of business world. On the business side, however, so much power remains in AI in everything from customer service and robotics to analytics and marketing. Companies using AI to surprise, connect, and communicate with their customers in ways they may not even appreciate or realize. This includes faster, cheaper, and smarter automation of everything in the competitive market.

In a competitive market, sellers make their products and services available, allowing the customers to select from a variety of options with the best prices. Simultaneously, the merchants should make sure to make sales without any loss, still retaining the customer base. Among the strategies to maintain a win-win situation are offering discounts and promotions. Hence, it is useful to create a payment platform that allows both the seller and the buyer to bargain online. In that type of a scenario, Multi Agent based solutions can provide with smart solutions.

Multi Agent based solutions (MAS) can be used for generating smart solutions in complex systems[4][5][6]. Multi Agent solutions are relatively new and limited applications have been reported in digital payment platforms. The researcher has identified the needs, challenges and opportunities in the market. So far, little progress has been made in developing a system incorporating bargaining facilities. This thesis has identified ordering product by bargaining in a digital payment platform as the research problem and proposed a Multi agent-based solution using Genetic algorithm when finding the most suitable merchant.

## 1.2 Aims and objectives

The aim of this study is to develop a web based or mobile based system option for putting an order and bring something through a platform in a secure manner by negotiating with the best merchants in the echo system..

In addition, following listed are the major objectives related to the research:

1. Critical review of digital payment platform challenges
2. In depth study of technology used for order products in payment platforms
3. Critically analyze applicability of Genetic algorithms when finding quick optimal solutions in dynamic environment.
4. Evaluate the solution
5. Writing a research paper and a conference paper on the project
6. Producing the final documentation

## 1.3 Background and Motivation

E-shopping has grown in popularity over the years, mainly because people find it convenient and easy to buy various items comfortably from their office or home. Software agent technologies provide a new scenario that is used to develop the new-generation e-commerce system, in which the most time-consuming stages of the customer's shopping process will be automated [2]. By the definition of ECA (Electronic Commerce Association) electronic commerce covers any form of business or administrative transaction or information exchange that is executed using any

information and communications technology [1]. Agent based e-commerce has emerged and become the focus of the next generation of e-commerce [3][4][7].
Agents have been claimed to be the next breakthrough in software development, resulting in powerful multi-agent platforms and innovative e-business applications. However, agent technologies have not achieved yet that level of maturity as required by advanced e-business applications [1][8].
As a team lead of such kind of project, identified the needs, challenges and opportunities in the market. Even though there are many web based and mobile based platforms in the market there is no such system to facilitate customers to order some product by negotiating the prices with the identified best merchants in the echo system using past data.

It is a possibility to request details of any product or service in a certain area. Taxi ride service apps which are widely in use across the country is one such instance. Nevertheless, that kind of simple solution cannot yield benefits to the customers. As a payment platform system can store more data of customers and merchants for a significantly longer period, it is viable to trace the patterns of customer and merchant behavior. The study discusses that use of AI technologies can facilitate customers to find the best merchants in an efficient way.

## 1.4 Problem in brief

Currently there are many digital payment platforms in the market and innovative features bring customer attraction and the more advantages in the competitive market to the platform owners.

Even though there are many challenges and opportunities in this filed as discussed in the above chapters, this research focus on the below business opportunity currently available in the market.

- A web based or mobile based system option for putting an order and bring something through a platform in a secure manner by negotiating with the best merchants in the echo system.

Merchants also will be benefited from this kind of solution to serve in a better way and increase their revenue in an easiest manner.

## 1.5 Proposed solution

For addressing such kind of problem which explained in the above, Multi-Agent based solution is proposing according to the study done in the previous days. An user interfaces going to add into application and the customers will be able to request some product using it.

Based on the requested product, customer location, his or her expenditure pattern and other considerable factors, this request will sent into the server and using Genetic algorithm the most suitable merchant will select and the request will send him to accept. The customer and merchant will be allowed to communicate with each other by negotiating the price through multi agent system and incase of rejecting the request another merchant will be proposed by the system or the current merchant. The security level of this kind of system must be into an industry standard level with zero bugs.

## 1.6 Resource Requirements

**Software:**

**Server side:**

- Windows 7 or above platform
- Jboss 6.2/Tomcat Application server
- MySQL database
- JDK 1.8
- Spring 3.0 or above
- Jade or similar framework
- JPA 2.0 or above

**Client side:**

- Windows 7 or above platform
- Chrome or other web browser

**Hardware:**

**Server side:**

- Computer/Laptop with Intel i5 Processor
- minimum 8GB RAM
- 200 GB or above hard drives (based on the stored data)

**Client side:**

- Normal windows Computer/Laptop with internet facility
- minimum 4GB RAM
- 200 GB or above hard drives

## 1.7 Structure of the thesis

This thesis has been structured with 8 chapters. Chapter 1 gave an overall introduction the project. Chapter 2 provides a critical review of the developments and issues in the area of digital payment platform by defining the research problem and identification of technologies. Chapter 3 is on technology adapted to building the multi agent solution for digital payment platforms. Chapter 4 presents our approach to multi-agent based solution for ordering product in digital payment platform. Chapter 5 is about the design of the proposed solution and it describes the modules and the connections. Chapter 6 presents the implementation details of each module and connections in the design. Chapter 7 gives the evaluation details such as experimental design, text cases, evaluation strategy and data correction. Finally Chapter 8 concludes the thesis with a note on the possible further work.

## 1.8 Summary

This chapter provided an introduction to the entire project. For this purpose, we have presented our research problem, objectives, technology use, proposed solution and resource requirements. Next chapter provides a detailed critical review of digital payment platforms.

Chapter 2

# Development and challenges in digital payment platforms

## 2.1 Introduction

Chapter 1 gave an introduction to overall project. This chapter presents a critical review of literature on digital payment platforms.  Here we formulate our research problem and highlight the technology adopted towards a solution.  In doing so, this chapter has been structured with four sections, namely, MAS in Industrial Landscape, gestation of digital payment platform, major developments, future trends and problem definition. Summary section contains a table which presents a summary of critical findings of the literature review.

## 2.2 MAS in Industrial Landscape

Among AI technologies, Multi-Agent Systems (MAS) are more popular in the industry. MAS have been widely implemented in business domains of manufacturing, electrical engineering, electronic, commerce, graphics (e.g., computer games and movies), transportation, logistics, software development, robotics, telecommunications and energy. Most of the giants in IT industry are using those technologies to develop commercial products to identify more opportunities in the competitive market.

MAS solutions are using since early 90s in the industry. MAS based solutions improves the operating efficiency, thus maximizing the power generation of the micro-grid (electrical engineering) and minimizing the cost of operation. The results of the effective communication sections clearly indicate that further negotiation has benefited for the significant change in the in supply chain management systems (manufacturing) which cannot achieve by other conventional technologies. The domain of traffic and transportation systems is ideal for an agent based approach because of its geographically distributed and dynamic changing nature. By using MAS it can build 3D game environments with self-organized 3D models positioned and oriented in most suitable places. It was proven that using MAS solution is highly suitable to access and distribute sensory data efficiently and timely manner in wireless sensor network environments. Likewise there are many industrial applications of MAS. However, the main challenge in the industrial application of multi-agent systems is to convince industry people regarding the benefits of using agents.

Multi-agent technology has been used in many areas but industry applications have taken the earliest advancement of agent technology when compared to others. The features of MAS technology can be simply used to handle the complexity in communication [9]. Multi-agent systems are derived from the distributed artificial intelligence and which allows an alternative way to design distributed control systems based on autonomous and cooperative agents, exhibiting modularity, robustness, flexibility and adaptability. MAS technology has proven how complex systems can be modeled to generate smart solutions, which could not be done easily using classical computing technologies even though they are little bit success in the recent history. Multi-agent technology has been used in many areas but industry applications have taken the earliest advancement of agent technology when compared to others [9].

MAS are using in many areas like production planning, inventory management, vehicle routing, electronics & electrical productions and software development industry [10]. A large volume of literature has shown the potential of multi-agent system (MAS) technology to effectively use the power of communication as a problem-solving strategy in complex systems. This chapter describes some of the industrial applications of MAS.

Many electrical engineering products are benefited from the advantages of this technology in many ways. Mini/Micro grids are one of the best applications of MAS in the electrical engineering world. Mini/micro grids are a potential solution being studied for systems relying on distributed generation. The basic idea behind the applicability of this is the communication of an agent that controls a production unit with its market environment. By using MAS, micro grids can be connected to the main power network or be operated autonomously, similar to power systems of physical islands. Implementing outage management using Multi Agent System of the micro-grid with smart grid frame can improve the efficiency in a considerable manner and solves a number of specific operational problems. This can be developed by accounting the intermittent nature of solar power, randomness of load, dynamic pricing of grid and discrepancy of critical loads and choose the best possible solution for the grid outage management in the micro-grid. Furthermore, MAS improves the operating efficiency, thus maximizing the power generation of the micro-grid and minimizing the cost of operation. Thus MAS in micro-grid leads to best use of monetary and environmental resources [11].

Supply chain management (SCM) can be considered as a well-known industrial application of MAS. Supply chain systems  has been developed as an agent enabling the essential features of MAS  (communication, coordination and negotiation)  among the agents to achieve intended business goals. The identified key roles in the supply chain are raw material suppliers, manufacturers, distributors and retailers. These agents are attached to different containers of the system. Agents have different behaviors and their decisions may be based on defined ontologies. The essential features of MAS technology can be simply used to handle the complexity in communication, which plays the vital role in SCM like in other areas. In fact, the results of the effective communication sections clearly indicate that, further negotiation has benefited for the significant change in the SCM which cannot achieve by other conventional technologies [9].

Agent technology is used in many real word applications in traffic management systems in dynamic environments to handle the complexity in an easiest manner. The domain of traffic and transportation systems is ideal for an agent based approach because of its geographically distributed and dynamic changing nature and with multi agent systems to enhance the ability of the traffic management systems to deal with the uncertainty in a dynamic environment [12].

OASIS is an agent based air traffic control system used in Sydney airport, Australia which was introduced in early nineties [9]. Which was developed to help alleviate air traffic congestion. The system achieves this by maximizing runway utilization, achieved through arranging landing aircraft into an optimal order, assigning them a landing time, and then monitoring the progress of each individual aircraft in real time. Its major components are independent agents, each solving a part of the overall problem like in other MAS. The figure 2.1 explains the architecture of the OASIS system [13].

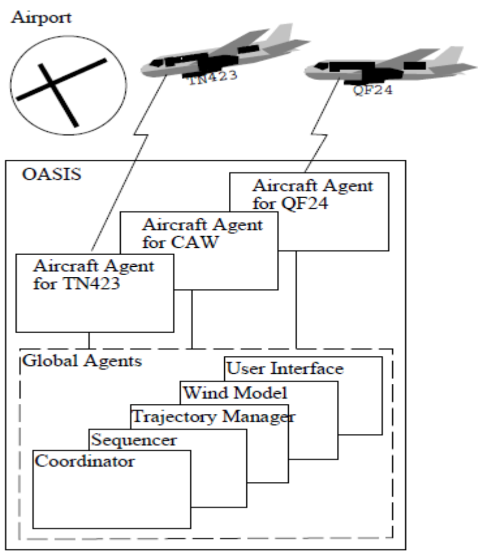


*Figure 2.1: Architecture of OASIS system*

OASIS is one of the best industrial application of MAS which was introduced at the very beginning and hope which was inspired by many of the other industrial applications in MAS.

One of the interesting areas in which MAS used is 3D games. Designing complex and reasonable 3D environments for modern 3D games is one of the time consuming challenges faced by present video game industry. Each 3D model in a 3D game environment can be associated with an agent with simple rules and system can allow users to introduce new 3D models and associate them with agent types. By using MAS it can build a 3D game environments with self-organized 3D models positioned and oriented in most suitable places [14].

MAS solutions can be applied in the wireless sensor network environments. It can be implemented by combining the use of intelligent sensors and middle agent architecture. Intelligent sensor nodes may exploit as autonomous agents which may monitor the incidents in the environment. For retrieving the data from the target nodes, mobile software agents can be used in an energy saving manner. It’s been proven that using MAS solution is highly suitable to access and distribute sensory data efficiently and timely manner [15].

With the development of software industry, software architectures and tools are also heavily influence the future trends in deployment of agent base solutions [16]. Agent based software developments has the potential to significantly improve the designing and implementing systems in disruptive manner. Agent based software systems are significantly popular for the purpose of solving complex real world problems. Especially, as MAS has the capability of being robust, scalable and working as autonomous agents who can achieve their objectives in an uncertain environment by communicating, coordinating and negotiation with each other [17].

## 2.3 Gestation of digital payment platform

Today's business environment is extremely competitive and globally connected. We are certain that multi-agent software is the correct approach to creating responsive and adaptable business solutions to meet the demands in this dynamic environment. There are lot of researches has been conducted to built powerful multi-agent platforms and innovative e-business applications in the recent history [1].

One of the most advantages of online shopping is saving our time especially during peak hours and a festival season. Otherwise we need to wait in long lines or search from store to store for a particular item. The unpredictable growth of the Internet users in world opened a new business opportunity to the whole world. Shopping activities over the internet have been growing in an exponential manner over the last few years. One of such environments in which there is a prominent job for the agents would be e-shopping in which a user is able to give those agents the responsibility of buying and selling, instead of searching the e-shopping himself. There are no proper mechanisms to facilitate electronic transaction and automate shopping process on behalf of customers. So a human buyer is still responsible for gathering commodity information from multiple suppliers on Internet, making decisions about each commodity, then making the best possible selection, and ultimately performing the e-payment. So it takes lot of time to buy things over the Internet. Hence, to reduce the time and to enhance the automation of the e-shopping system a multi agent environment is used [5].

New payment disrupters like  Google Wallet, Dwolla, are in attempt to change the status quo of payment. By issuing new digital payment instruments, payment disrupters have the deliberate goal to challenge payment incumbents. But before a new payment instrument can be adopted, market players such as banks, acquirers, payment solution providers, mobile network operators (MNO); as well as merchants and cardholders need to be convinced. Mobile phone is gaining a foothold in the payment industry, how do payment disrupters, strategically design and manage their digital payment service, in order to be adopted on the payer and payee [19].

Payment systems have not received extensive attention as a research topic over the past decade given the relative stability and well-defined roles that exist in the industry, but recently they have attracted growing attention. Scholars have studied payment systems and their corresponding payment cards as so-called two-sided platforms (or, in general, multi-sided markets) that need to attract both merchants and cardholders to be viable [19].

The following figure 2.2 explain the basic design of a Digital Payment Framework

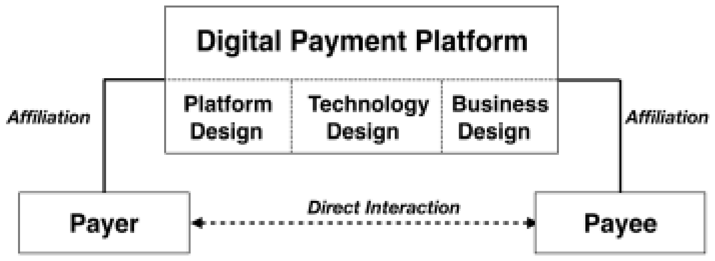


*Figure 2.2: Digital Payment Platform Design Framework*

| **Component** | **Description** |
|---|---|
| Direct Interaction | Classifies a platform as being a Multi Sided Platform |
| Platform Design | Describes open and closed systems, and how complementary products are distributed. |
| Technology Design | The applied technology based on evolutionary or revolutionary hardware strategies |

|  |  |
|---|---|
| Business Design | Market-entry strategies through bundling products and leveraging an installed user base (envelopment attack). Alternatively, through Schumpeterian innovation, which is more radical, but rare to achieve. |

*Table 2.1: Component description*

Digital platforms drive many markets, such as the payment market. Digital platforms are layered, modular technology artifacts that have the logic to match different users like payers and payees to derive business value. Because these layered, modular IT artifacts create value through mediation, digital platforms are considerably sophisticated in their technology attributes. Contemporary digital platforms like PayPal are equipped with application programming interfaces (APIs), which are access and distribution points for internally or externally developed services. Furthermore, digital platforms deliver services increasingly through physical means (e.g., mobile phones), which, in essence, represent physical proxies of digital platforms. Take PayPal as an example of a digital payment platform owner: PayPal offers APIs to third parties to integrate payment functionalities into their own mobile services [20].

Based on the abovementioned observations, payment platforms comprise various components (e.g., APIs and mobile phones) in delivering their services. Accordingly, to support conditions for platform envelopment, one has to accordingly design and configure platforms and their corresponding components in the first place. Platform envelopment, however, is a complex task and novel for some prior protected markets, such as the traditional payment market. As new payment actors with different industry backgrounds encroach the payment market and, thereby, disturb market equilibrium, established payment actors in their core markets are compelled to respond to remain relevant. To shed light on platform envelopment in the payment market, we study and explain how digital platforms leverage payment services as a mean to enter other existing platform markets [20].

The development of the Internet and the arrival of e-commerce fostered digitalization in the payment processes by providing a variety of electronic payment options including payment cards (credit and debit), digital and mobile wallets, electronic cash, contactless payment methods etc. Mobile payment services with their increasing popularity are presently under the phase of transition, heading towards a promising future of tentative possibilities along with the innovation in technology [21].

E-commerce has become a rapidly growing market today. With the proliferation of tablets and smart phones, the use of electronic payment methods has grown up to 21% in 2012. The use of credit cards was the major international means of online payment that dominated in a variety of transaction markets. It was estimated that 95% of all e-commerce transactions in the United States are performed using credit cards. Other widely used online payment alternatives are debit cards (with rising number of users worldwide) and online payment systems like Paypal, Stripe or Skrill. With the availability of a variety of electronic payment means including mobile payments, mediating services, and electronic currency, an appropriate option can be chosen for a particular type of transaction [21].

There are many different shopping sites on the Internet now; however, most of these sites lack a user-friendly interface design, which is essential for the success of online software [5]. The interface of e-shopping systems must be pleasing to the eye, effortless to learn and easy to use [5]. Otherwise, people in general will likely become less interested in e-shopping applications.

The Figure 2.3 explains the Entity Interaction for an E-Cash Payment Transaction [3].

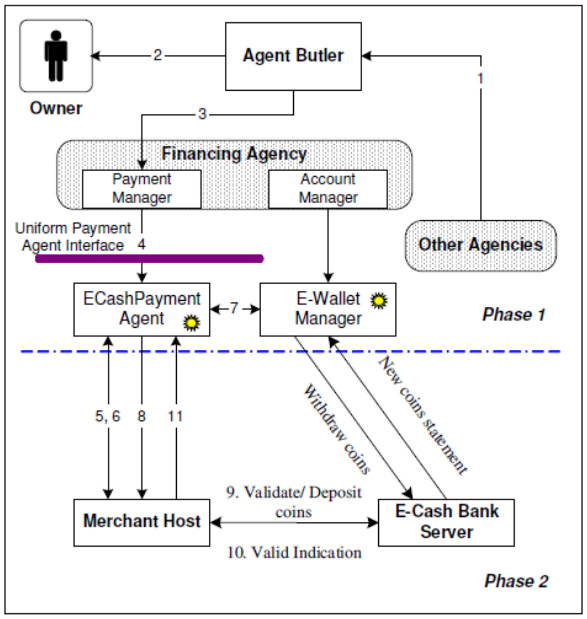


*Figure 2.3: Entity Interaction diagram for an E-Cash Payment Transaction*

In order to be widely accepted payment method across the global, electronic payment systems have to follow an efficient security protocol that must ensure a high security for online transactions. Two common protocols are identified that ensure secure e-commerce transactions. These protocols include Security Socket Layer Protocol (SSL) and Secure Electronic Transaction (SET). SSL is more commonly used e-commerce transactions protocol and it works by encoding the entire session amongst computers so that it enables to provide a safer communication over the internet. SSL encrypts the online communication between Web servers and a client by using public-key technology. On the other hand, SET protocol works by preventing consumer's entire credit card number from travelling across the internet instead allows pieces of it to flow through web communication. SET also offers information integration, coding of sensitive information, and verification of all business data by using latest technologies such as digital signature and data coding. And also we should consider about the following points as well;

- Confidentiality of information shared by consumers
- Data integrity
- Authentication of all the participants
- Non-repudiation
- End-user requirements that include usability, flexibility, affordability, reliability, speed of transactions, and availability

The introduction of mobile payments has risen several security issues by itself like cloning a device, app malware, identity theft and so on. On the other hand the device itself can also help add extra layers of security in the payment like tokenization, device and sim authentication, location patterns, user authentication including fingerprint authentication [21].

### 2.4 Future Trends

However, it is currently appreciated that for various reasons agent technologies have not achieved yet that level of maturity as required by advanced e-business applications. Many difficult research challenges remain, and much work is needed to adapt relevant existing agent technologies to the requirements of the new generation ebusiness systems. Some frameworks have arisen that implement common standards (such as the FIPA agent system platforms and communication languages). These frameworks save developers time [1].

There are many future trends in this area. An intelligent shopping system can be developed using a multi-agent system to provide shopping service for the commodities that a consumer does not buy frequently. The system integrates built-in expert knowledge and the customer's current needs, and recommends optimal products based on multi-attribute decision making method [5].
Software agent technologies provide a new scenario that is used to develop the new-generation e-commerce system, in which the most time-consuming stages of the customers’ shopping process will be automated [5].

At now on the internet most of the e-shopping systems are using the normal webpage implementation. That will take the more heavy work for the web servers. Intelligent e-shopping systems can be developed using data mining. But it can be too slow. The agent technology can be used to reduce the enormous time taken by the e-shopping system based on web application and data mining [5].

## 2.5 Problem definition

The exponential development of the Internet has changed the way enterprises do business. Electronic commerce is becoming an attractive means for conducting business transactions. However, the progress of e-commerce seems to be hindered by the lack of a widely accepted payment standard suitable for e-commerce.
Lack of intelligence is another factor stymieing electronic commerce is also emerging to the surface. The vast size of information on the Internet also means that it is difficult for potential customers to locate products that they are interested in. Therefore, e-commerce demands advanced technologies as support. Agent technology seems to be an excellent candidate with its properties of intelligence, autonomy, and mobility [3].

## 2.6 Summary

The agent technology is used to enhance the customer’s needs which include availability, speedy response time, and efficiency. Agent for e-Shopping creates connectivity on an anytime anywhereany-device-basis to provide the specific goods required by the consumers based on transaction cost optimization and scalability. Agent based e-commerce has emerged and become the focus of the next generation of e-commerce. The following table contains a summary of a literature review.

| **Summary** | **Reference** |
|---|---|
| Modern business environment is dynamic and constantly changing, however customers' expectations are high | T.V. Solodukha, O.A. Sosnovskiy), B. A. Zhelezko "Multi-Agent Systems for E-Commerce". |
| MAS can be used for generating smart solutions in this kind of complex systems | M. C. G. Bih-Ru Lea, "A prototype multi-agent ERP system." Y. P. T. Finin, "A Multi-Agent System for Enterprise Integration." |
| The most time-consuming stages of the customer's shopping process is automated using MAS | S. K. Zhang Yuheng, "A MULTI AGENT BASED E-SHOPPING SYSTEM," Volume 2, No. 4, April 2011 Journal of Global Research in Computer Science. |
| Agent based e-commerce has emerged and become the focus of the next generation of e-commerce | S.-U. G. Feng Hua, "A Multi-Agent Architecture for Electronic payment," International Journal of Information Technology and Decision Making · November 2011 |
| Agents have been claimed to be the next breakthrough in software development, resulting in powerful multi-agent platforms and innovative e-business applications | T. V. S. O.A. Sosnovskiy, "Multi-Agent Systems for E-Commerce". |
| Agent technologies have not achieved yet that level of maturity as required by advanced e-business applications | Paulo Leitao, "Multi-agent Systems in Industry:Current Trends & Future Challenges."Online].Available:https://pdfs.semanticscholar.org/234c/6a74f071697baf3f8211ba5244e111971850 .pdf |

| | |
|---|---|
| An intelligent shopping system can be developed using a multi-agent system to provide shopping service for the commodities that a consumer does not buy frequently | S. K. Zhang Yuheng, “A multi agent based e-shopping system,” Volume 2, No. 4, April 2011 Journal of Global Research in Computer Science. |
| MAS technologies provide a new scenario to develop the new-generation e-commerce system, in which the most time-consuming stages of the customers’ shopping process will be automated | S. K. Zhang Yuheng, “A multi agent based e-shopping system,” Volume 2, No. 4, April 2011 Journal of Global Rese |

*Table 2.2: Summary of Literature Review*

Chapter 3

# Technology adopted - Multi Agent and Genetic Algorithm

## 3.1 Introduction

Chapter 2 formulated our research problem and highlights the technology adopted towards a solution. Mainly three technologies were identified to use in this project. Multi Agent System technology is essential for communicating, coordinating and negotiating with merchant and customer. Based on the requested product, customer location, his or her expenditure pattern and other considerable factors, system needs to find the most suitable merchant to facilitate the customer. According to the literature review, Genetic algorithm can be used for this to find a solution in an efficient manner. This chapter has been structured with five sections, namely, Multi Agent System technology, Genetic Algorithm technology, Java Persistence API, Spring MVC framework and MySQL. Multi Agent System technology section is the main section of this chapter and it contains the frameworks which can be used for developing this kind of system.

## 3.2 Multi Agent System technology

The modern business environment is dynamic and constantly changing, but customers expect their specific requirements to be met at the agreed service level. The Internet and mobile computing means that the majority of business processes are "always-on" and increasingly interconnected. Agent based solutions are designed to handle complex and changing environments, where there are potentially multiple options and outcomes at each stage of the process. Multi-agent systems can offer measurable business benefit for many organizations [1].

The study of multi-agent systems focuses on systems in which many intelligent agents interact with each other. Agents are software programs running on a system or network and implemented to achieve a goal or purpose. They are designed to be autonomous in the pursuit of their goal, but capable of interacting with other agents, either collaboratively or in competition. This means that groups of agents can work together, each carrying out their own plans and functions. This provides a powerful technique for designing software applications for complex or distributed business processes [1].

Several researchers have attempted to provide a meaningful classification of the attributes that agents might have. A list of common agent attributes is shown below [17].

**Adaptivity**: the ability to learn and improve with experience.

**Autonomy**: the ability to act without any interference from the outside

**Activity**: the ability to show its own initiative.

**Collaborative behavior**: the ability to work with other agents to achieve a common goal.

**Mobility**: the ability to migrate in a self-directed way from one host platform to another.

**Reactivity**: the ability to selectively sense and act.

**Temporal continuity**: persistence of identity and state over long periods of time.

Multi-agent systems are designed to handle changing and dynamic business processes. They continue to operate even when there may be incomplete information, but a decision must still be made by the most effective method. Properly integrated into a business process they can offer flexible and intelligently adaptable systems support [1].

### 3.2.1 MaSMT framework

MaSMT is a free, java based lightweight Multi-agent system development framework. It provides three types of agents, as ordinary agent and managing agent and root agent. The managing agent capable to handle set of ordinary agent and the root agent capable to handle set of manager agents. MaSMT3.0 includes set of new features other than the previous versions. It includes root agent to handle swam of agents, Environment handling features to dynamically store agent's ontology, and notice board has been introducing to see required messages and events. In addition to these main features, agent status monitor has been introducing to view transporting messages.

### 3.2.2 JADE

JADE (Java Agent DEvelopment Framework) is a Java base software Framework which simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications( IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies) and through a set of graphical tools that support the debugging and deployment phases. A JADE-based system can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required.

## 3.3 Genetic Algorithm Technology

Genetic algorithms (GAs) are a heuristic search and optimization technique inspired by natural evolution. They have been successfully applied to a wide range of real-world problems of significant complexity. Starting with a randomly generated population of chromosomes, a GA carries out a process of fitness based selection and recombination to produce a successor population, the next generation. During recombination, parent chromosomes are selected and their genetic material is recombined to produce child chromosomes. These then pass into the successor population. As this process is iterated, a sequence of successive generations evolves and the average fitness of the chromosomes tends to increase until some stopping criterion is reached. In this way, a GA "evolves" a best solution to a given problem [18].

GAs was first proposed by John Holland as a means to find good solutions to problems that were otherwise computationally intractable. Holland's Schema Theorem, and the related building block hypothesis, provided a theoretical and conceptual basis for the design of efficient GAs. As a consequence, the field grew quickly and the technique was successfully applied to a wide range of practical problems in science, engineering and industry [18].
They are particularly suited to problems where traditional optimization techniques break down, either due to the irregular structure of the search space (for example, absence of gradient

information) or because the search becomes computationally intractable [18].There are lot of research has been conducted in this area and proven that GA performs better than other local search methods [22].

## 3.4 Java Persistence API (JPA)

Java Persistence API (JPA) is a Java application programming interface specification that describes the management of relational data in applications using Java Platform. JPA 2.0 was started in July 2007 in the Java Community Process as JSR 317 but it was approved in December 2009.

Main features included were:

- Expanded object-relational mapping functionality
    - support for collections of embedded objects, linked in the ORM with a many-to-one relationship
    - ordered lists
    - combinations of access types
- A criteria query API
- standardization of SQL Hints
- standardization of additional metadata to support DDL generation
- support for validation
- Shared object cache support

## 3.5 Spring MVC framework

The Spring MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

- The **Model** encapsulates the application data and in general they will consist of POJO.

- The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
- The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

The Spring Web model-view-controller (MVC) framework is designed around a “DispatcherServlet” that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC “DispatcherServlet” is illustrated in the following figure 3.1.

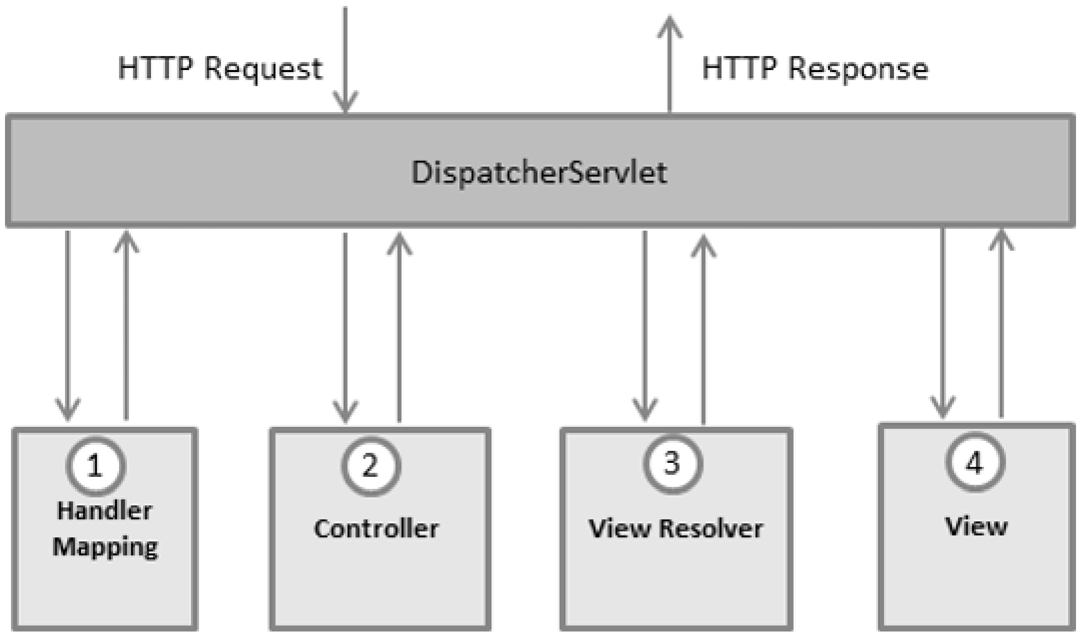


*Figure 3.1: Spring MVC architecture*

Following is the sequence of events corresponding to an incoming HTTP request to “DispatcherServlet”

- After receiving an HTTP request, “DispatcherServlet” consults the HandlerMapping to call the appropriate *Controller*.
- The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the “DispatcherServlet”.

- The *"*DispatcherServlet*"* will take help from ViewResolver to pickup the defined view for the request.
- Once view is finalized, The *"*DispatcherServlet*"* passes the model data to the view which is finally rendered on the browser

Following are the few major advantages of Spring Framework

1. Spring enables the developers to develop enterprise applications using POJOs (Plain Old Java Object). The benefit of developing the applications using POJO is, that we do not need to have an enterprise container such as an application server but we have the option of using a robust servlet container.
2. Spring provides an abstraction layer on existing technologies like servlets, jsps, jdbc, jndi, rmi, jms and Java mail etc., to simplify the development process.
3. Spring comes with some of the existing technologies like ORM framework, logging framework, J2EE and JDK Timers etc, Hence we don't need to integrate explicitly those technologies.
4. Spring WEB framework has a well-designed web MVC framework, which provides a great alternate to legacy web framework.
5. Spring can eliminate the creation of the singleton and factory classes.
6. Spring provides a consistent transaction management interface that can scale down to a local transaction and scale up to global transactions (using JTA).
7. Spring gives built in middleware services like Connection pooling, Transaction management and etc.,
8. Spring provides a light weight container which can be activated without using webserver or application server.

## 3.6 MySQL

MySQL is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language **(**SQL**).**

SQL is the most popular language for adding, accessing and managing content in a database. It is most noted for its quick processing, proven reliability, ease and flexibility of use.

Features of MySQL Database:

- Easy to use: MySQL is easy to use. You have to get only the basic knowledge of SQL. You can build and interact with MySQL with only a few simple SQL statements.
- It is secure: consist of a solid data security layer that protects sensitive data from intruders. Passwords are encrypted in MySQL.
- Client/ Server Architecture: follows a client /server architecture. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they query data, save changes, etc.
- Free to download: MySQL is free to use and you can download it from MySQL official website.
- It is scalable: MySQL can handle almost any amount of data, up to as much as 50 million rows or more. The default file size limit is about 4 GB. However, you can increase this number to a theoretical limit of 8 TB of data.
- Allows roll-back: allows transactions to be rolled back, commit and crash recovery.
- High Performance: MySQL is faster, more reliable and cheaper because of its unique storage engine architecture.
- High Flexibility: supports a large number of embedded applications which makes MySQL very flexible.
- High Productivity: MySQL uses Triggers, Stored procedures and views which allows the developer to give a higher productivity.

Following are the few disadvantages of MySQL:

- MySQL version less than 5.0 doesn't support ROLE, COMMIT and stored procedure.
- MySQL does not support a very large database size as efficiently.

- MySQL doesn't handle transactions very efficiently and it is prone to data corruption.
- MySQL is accused that it doesn't have a good developing and debugging tool compared to paid databases.

## 3.7 Summary

This chapter provided detail description about the technologies use throughout this project and the explanation for using those technologies. MAS based solution is ideal for this kind of complex system development and it handles the complexity of negotiation, coordination and communication of the dynamic environment. Agents can be designed to be autonomous in the pursuit of their goal, but capable of interacting with other agents, collaboratively. And also it was identified that JADE framework is more suitable for developing MAS based solution using JAVA language. In Next chapter provides a detailed explanation of the approach of this project and it's the most critical section of this thesis.

Chapter 4

# Novel approach to customer driven bargain

## 4.1 Introduction

Chapter 3 presented a broad overview of Technologies use throughout the project. This chapter presents the novel approach of this project and it will give visualization about the entire project. As such we have structure the chapter with subsections, namely, hypothesis, process, input of the system, outputs of the system, overall features of the system and users of the system. In the description, the section on process gives overall functionalities together with the relevant technologies.

## 4.2 Hypothesis

In our research we have hypothesized that the issue of finding a web based or mobile based system option for putting an order and bring something through a platform in a secure manner by negotiating with the best merchants in the echo system. Our thinking has been inspired by the power of MA technology for solving distributed complex systems.

## 4.3 Process

The MAS for digital payment platform has been designed by assigning an Agent for each distribution point. These agents are able to communicate, negotiate and coordinate when they deliberate for solution. These activities take places autonomously. The MAS solution also involves a manager agent who facilitates initializing agents, resource allocation, notification of agent, etc . Our new approach to customer bargaining is named as MASCB acronyms for Multi Agent System for Customer Bargaining. Top level architecture of our proposed MASCB system is shown in Figure 4.1.

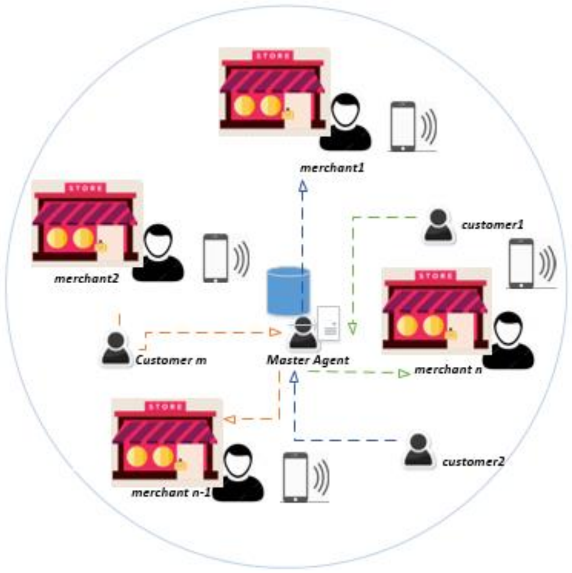


*Figure 4.1: merchant customer interaction top level Architecture diagram*

Based on the requested product, customer location, his or her expenditure pattern and other considerable factors, this request will sent to the search agent for selecting suitable merchants in that area. Here the search agent is using a Genetic algorithm process and distance between the merchant and the customer, previous customer reviews for the particular merchant is considered in the fitness function. That process with propose set of merchants who satisfy the fitness function and customer agent send a request to all these merchant for offering a price for the particular product. Any desired is allowed to offering a price that customer and the merchant who sends the minimum price will introduce for the customer for accepting it. The customer and merchant will be allowed to communicate with each other by negotiating the price through multi agent system and incase of rejecting the request another merchant will be proposed by the system and above process will repeat until customer find the suitable merchant who satisfy with his/her requirements. If the customer accept that order it will inform to the merchant and the merchant will pass that order to the delivery module. This will open another business opportunity as well. System can be modified to host an API for any third party delivery partner to process the delivery process.

## 4.4 Input of the system

Agents in Digital payment platform receive many inputs from manager agent, fellow agents and human agents inputs by customer, merchant.

These are the inputs to the system.

- Product needs to order
- Product category
- Location of the customer
- Expect price
- Customer identification(customer ID)

By identifying the customer, system will be able to predict more information using the past data keeps in the echo system.

## 4.5 Output of the system

The following items are the main outputs of the system but there are many indirect advantages like time saving of customers and increase the sales of merchants.

- Introduce the most suitable merchant
- Order with the negotiated price
- Reorder option from another merchant

## 4.6 Overall features of the system

The following items are the main features implement from this project.

- Analyze the past data and select the most suitable merchant in the area
- Facilitate the customer and merchant to negotiate for prices
- Accurate
- Time saving

Apart from these key features there are lots of advantages of this developed system. Customers will be able to save their precious time in this modern complex world while saving their money with the help of this system.

## 4.7 Users

These are the main users of this system.

- Customers in a payment platform
  Here the main role of customer is to purchase or order product through this system. Customer needs to enter the product name with the category. Customer allows accepting merchant offer or request for new price or request order from another merchant

- Registered merchants in a  payment platform
  The main role of merchant is to give an offer to the customer for the requested order item or reject it. Likewise merchant has to accept or reject a request which customer send for reducing the price. Properly maintaining the inventory is also a role of the merchant and it effects to the accuracy of the system also.

## 4.8 Summary

This chapter provided a detailed explanation of the approach of this project and this is the most critical section of this thesis. It gives a detail explanation of the scope of this project and gives an idea about what kind of project is going to develop. It explained about the features of this system, inputs and the users and their role as well. Next chapter describes the design of the system implement from this project.

# Chapter 5

# Design of MASCB

## 5.1 Introduction

Chapter 4 explained the novel approach of this project and it was given visualization about the entire project. This chapter presents the Top level architecture of putting an order in a digital payment platform. It contains agents, who access a common ontology of knowledge of payment platform rule and practices. This chapter has been structured with three main sections, namely, Agent Communication, Architecture of the system and Database design. The main outcome of this chapter is the architecture of this system.

## 5.2 Agent Communication

The following diagram in figure 5.1 illustrates the agent communication between each other. The agents of four main categories communicate each other by passing messages.

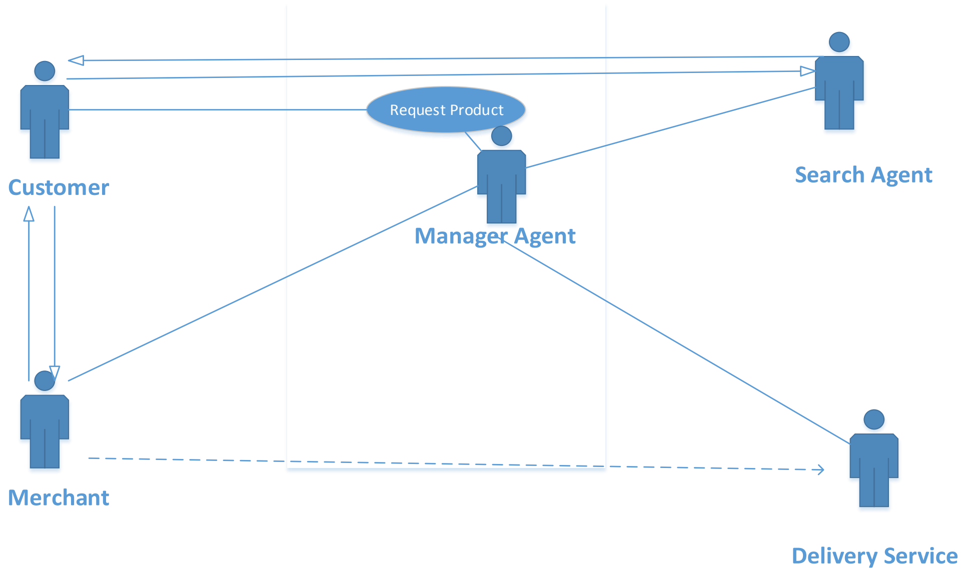


*Figure 5.1: Agent interaction diagram*

The flow chart in following figure 5.2 presents the overall flow of the MASCB.

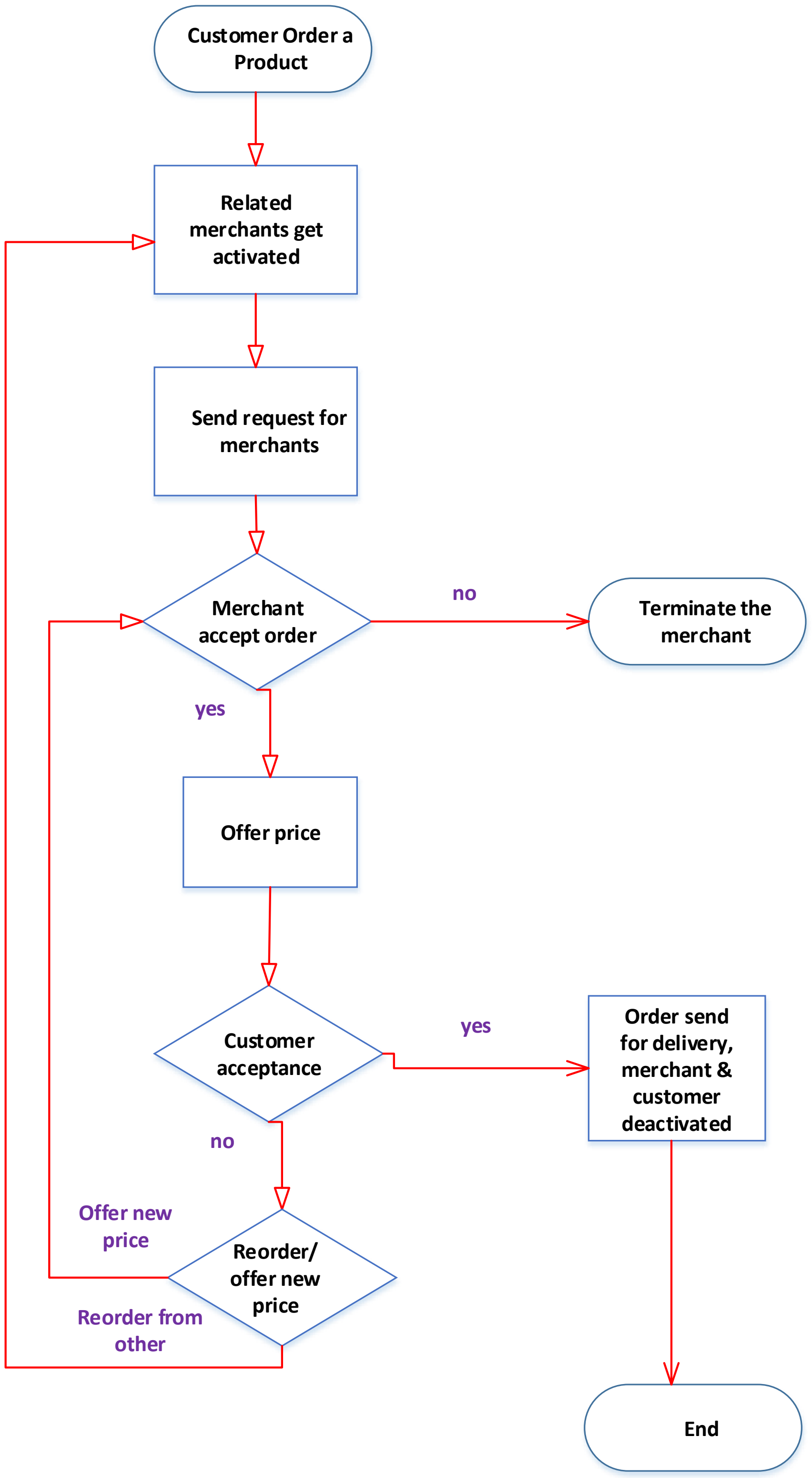


*Figure 5.2: MASCB workflow diagram*

The sequence diagram in Figure 5.3 provides the processes and objects involve, and the messages exchanged between them to perform.

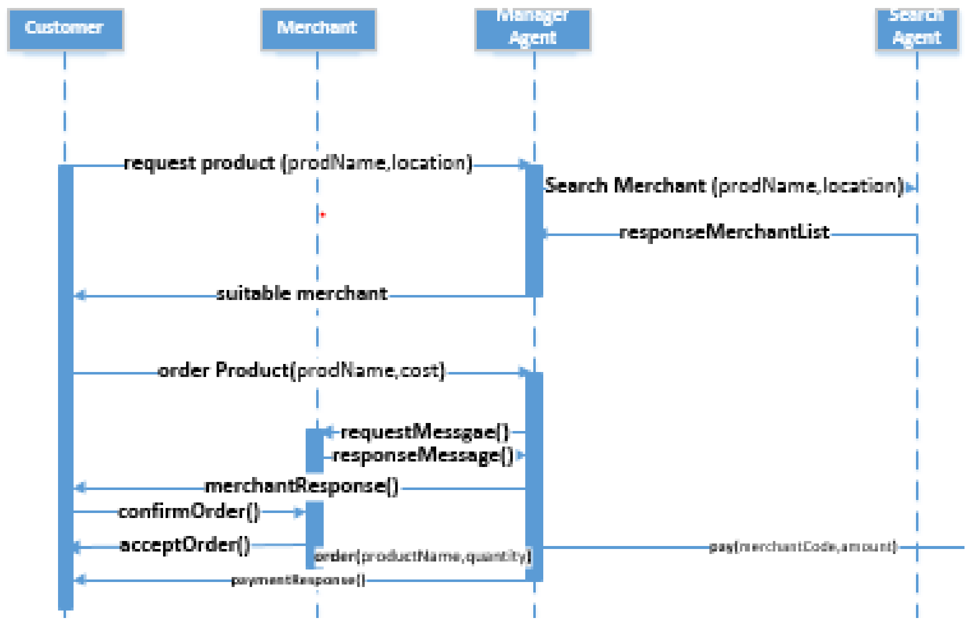


*Figure 5.3: Agent interaction sequence diagram*

Describe the following agents by indicating their roles/task and their connection with the other agents.

### 5.2.1 Manager Agent

This agent can communicate and even interrupt the other agents at his wish. This does happen with other agents unless in very exceptional situations. This agent is doing the main coordination part of the whole process. The main role of this agent is initialize agents, support execution, suspend and terminate the other agents.

### 5.2.2 Customer Agent

This agent represents the customer who initiates the request for ordering product. Agent needs to communicate with manager agent and has to communicate with the merchant agent for negotiating prices. These are the main functionalities of this agent.

- Order product
- Negotiate prices with merchants
- Reorder from another merchants

### 5.2.3 Merchant Agent

Merchant agent is another key agent in this system. This agent will communicate with the manager agent to make success this process. The following things are the main functionalities of  this agent.

- Accept customer order
- Offer price for the product
- Reject customer order

### 5.2.4 Search Agent

The responsibility of this agent is to find the more suitable merchants related to the product request sent by the customer. In this case search agent has to work with manager agent and customer agent to check whether the results satisfy the system defines constraints.

## 5.3 Architecture of the system

The following diagram 5.4 illustrates the architecture of this system.

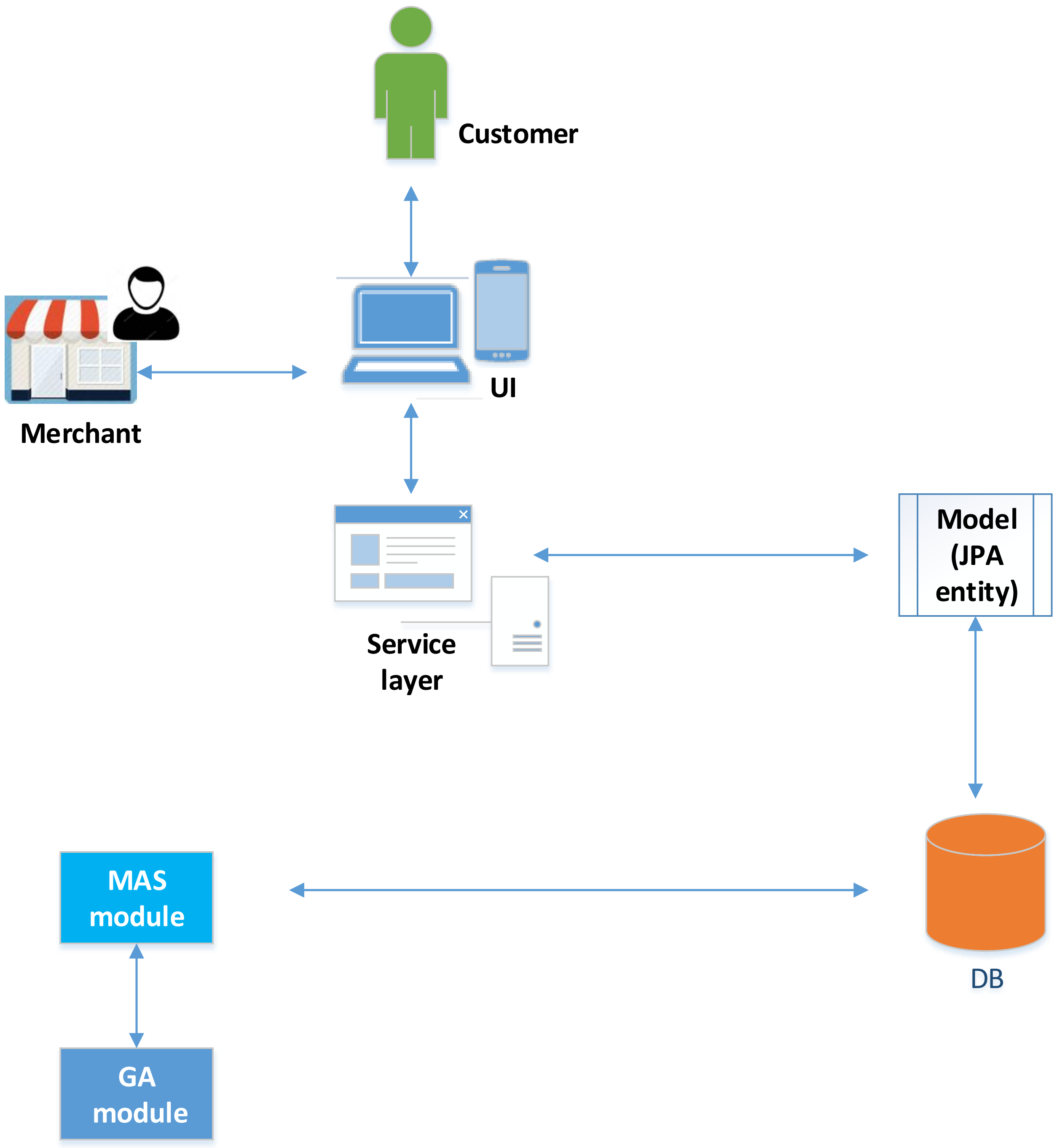


*Figure 5.4: Architecture diagram*

Mainly it contains UI layer, service layer, DB layer, MAS module and GA layer. The connection between each layer/module has presented in this diagram.

## 5.4 Database design

There are set of new tables need to create for developing this proposed module. Customer order storing table is the main table of this module and there are set of other tables to be integrated with this table. The ER diagram in the Figure 5.5 explains the DB design of this system.

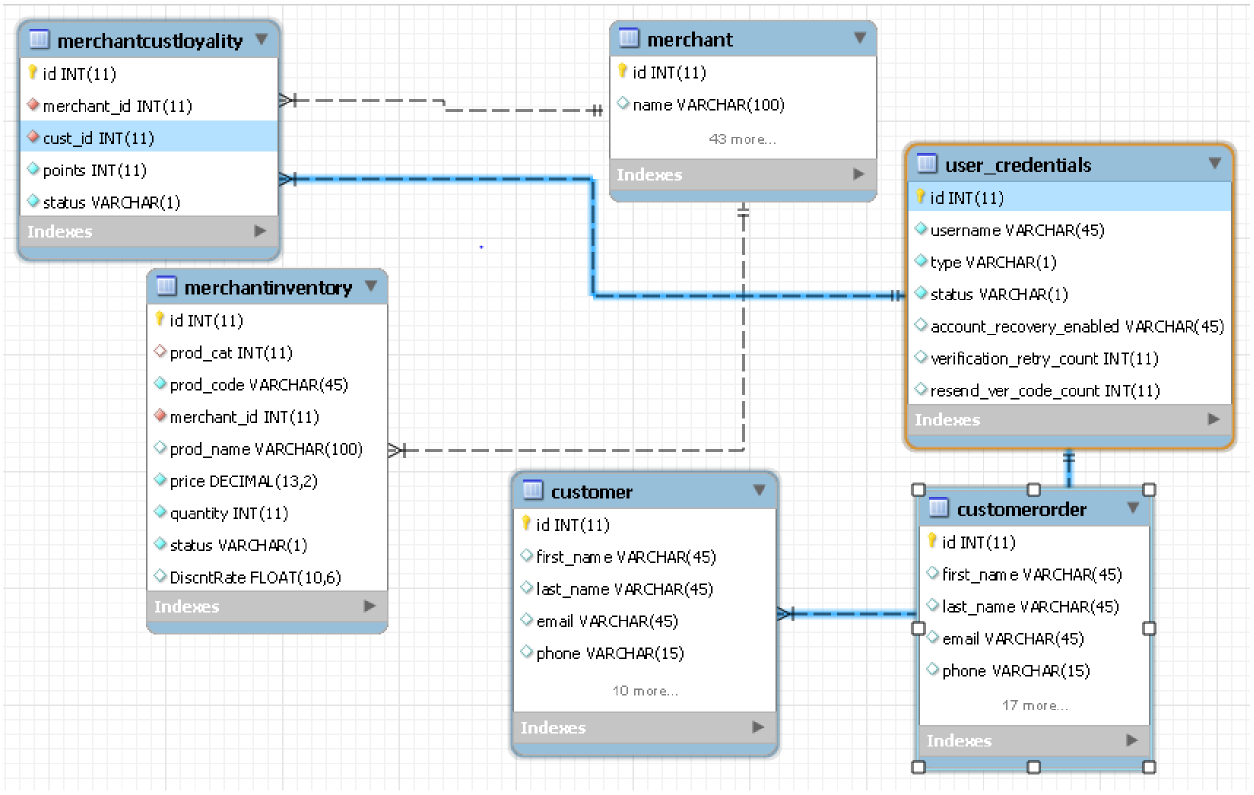


*Figure 5.5: ER diagram*

## 5.5 Summary

This chapter provided detail description of the design of the proposed solution and it describes the modules and the connections to the each other. The main outcome of this module is the architecture diagram of this system and it is used for the implementation process of this system.  This section is more important as the blueprint of the system development decides here and the whole system develop based on the decisions taken here. Next chapter presents the implementation details of each module and connections in the design.

Chapter 6

# Implementation

## 6.1 Introduction

Chapter 5 described the design of the entire system and this chapter presents the implementation of the proposed system. Implementation process mainly consists of four sections as Multi Agent System module, Genetic Algorithm process, DB module, API module and UI module. Genetic algorithm process which is used for selecting the best merchants include in the Appendix 2. The source code used in API module and the screens in UI module contains in Appendix 3 and Appendix 4. As such we have structure the chapter with subsections, namely, Agent development and pseudo code, Database Implementation, Technology and UI Implementation.

## 6.2 Agent development and pseudo code

The following pseudo codes and flow charts of the agent types explain the implementation of MAS module and which is the core module of this system.

### 6.2.1 Customer Agent

The flow chart in figure 6.1 illustrates the functionality of customer agent and the pseudo code explains how we are going to develop this agent in JADE framework.

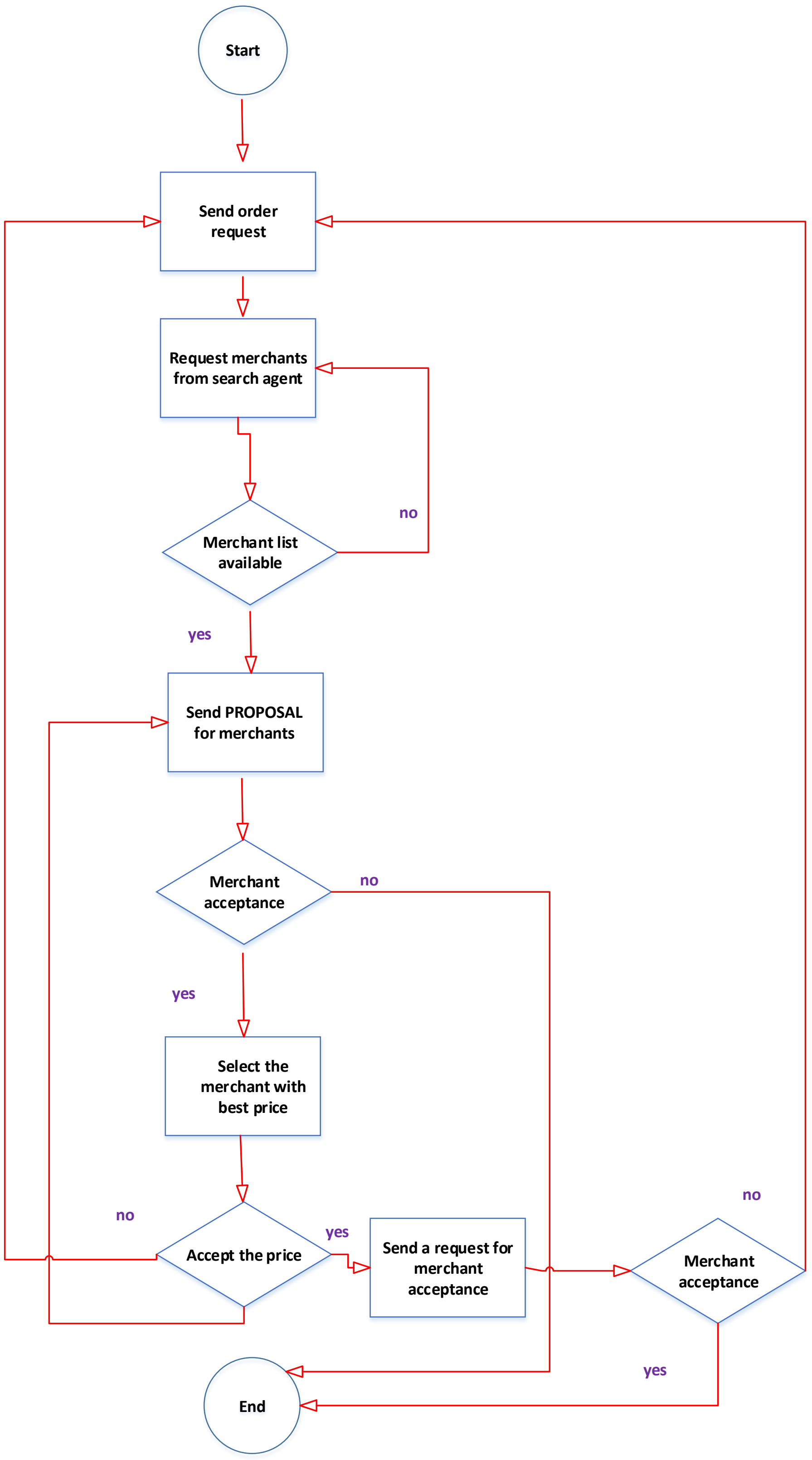

Start
Send order request
Request merchants from search agent
Merchant list available
no
yes
Send PROPOSAL for merchants
Merchant acceptance
no
yes
Select the merchant with best price
Accept the price
no
yes
Send a request for merchant acceptance
Merchant acceptance
no
yes
End

*Figure 6.1: Customer agent flow chart diagram*

```
AGENT  INITIALIZATIONS

START SETUP

        ADD BEHAVIOR RequestMerchant

END SETUP


// This is the behaviour used by customer agents to request product from merchant
agents
START BEHAVIOR RequestMerchant

        Send the CFP to Search Agent

        Receive the list of suitable merchants

        Send the CFP to suitable merchants

        Receive all proposals/refusals from merchant agents

        Send the purchase order to the merchant that provided the best offer

        Receive the purchase order reply

END BEHAVIOR
```

6.2.2 Merchant Agent

The flow chart in figure 6.2 illustrates the functionality of merchant agent and the pseudo code explains how we are going to develop this agent in JADE framework.

```
AGENT  INITIALIZATIONS

START SETUP

// Add the behaviour serving queries from buyer agents
                addBehaviour(new OfferRequestsServer());

// Add the behaviour serving purchase orders from buyer agents
                addBehaviour(new PurchaseOrdersServer());

END SETUP

START BEHAVIOR OfferRequestsServer
```

```
        Receive CFP messages from customers
        If requested item available
            ACLMessage.PROPOSE
        Else
            ACLMessage.REFUSE
END BEHAVIOR
```

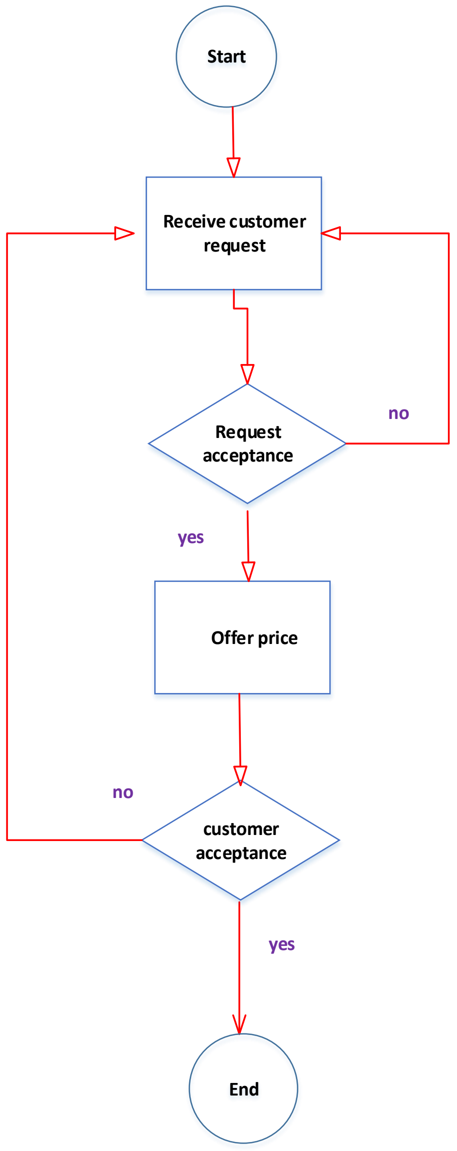


*Figure 6.2: Merchant agent flow chart diagram*

### 6.2.3 Manager Agent

The flow chart in figure 6.3 illustrates the functionality of manager agent and the pseudo code explains how we are going to develop this agent in JADE framework.

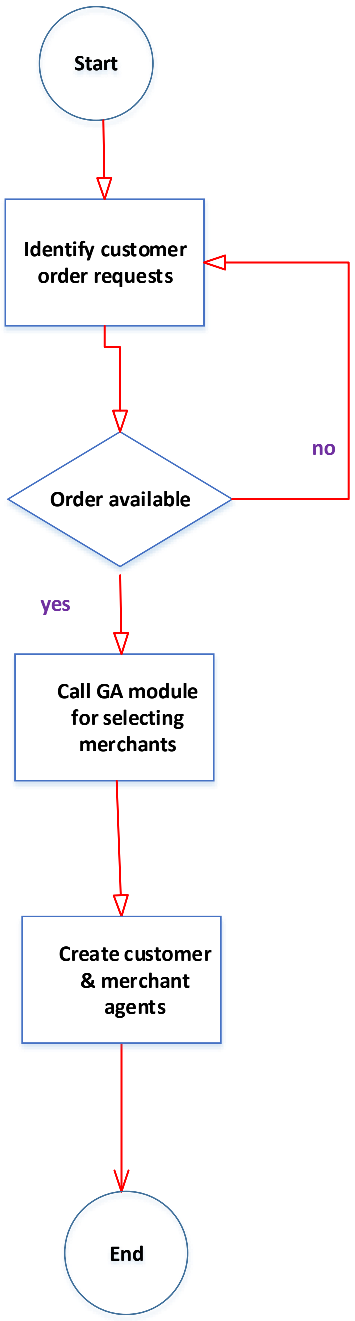

Start
Identify customer order requests
Order available
no
yes
Call GA module for selecting merchants
Create customer & merchant agents
End

*Figure 6.3: Manager agent flow chart diagram*

```
START SETUP
    Create customer agents
    Create merchant agents
END SETUP
```

### 6.2.4 Search Agent

The flow chart in figure 6.4 illustrates the functionality of search agent and the pseudo code explains how we are going to develop this agent in JADE framework.

```
AGENT  INITIALIZATIONS

START SETUP

// Add the behaviour serving queries from customer agents
                addBehaviour(new proposeMerchants());
END SETUP

START BEHAVIOR proposeMerchants

        Receive CFP messages from customers

        If suitable merchants available

            ACLMessage.PROPOSE

            Send the merchant list

        Else

          ACLMessage.REFUSE

END BEHAVIOR
```

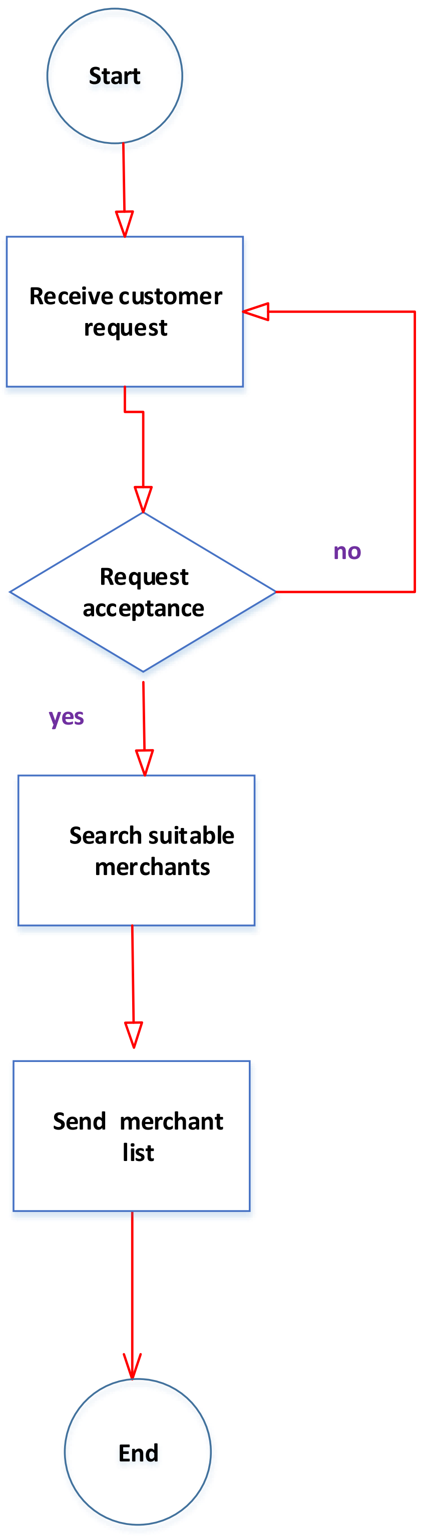


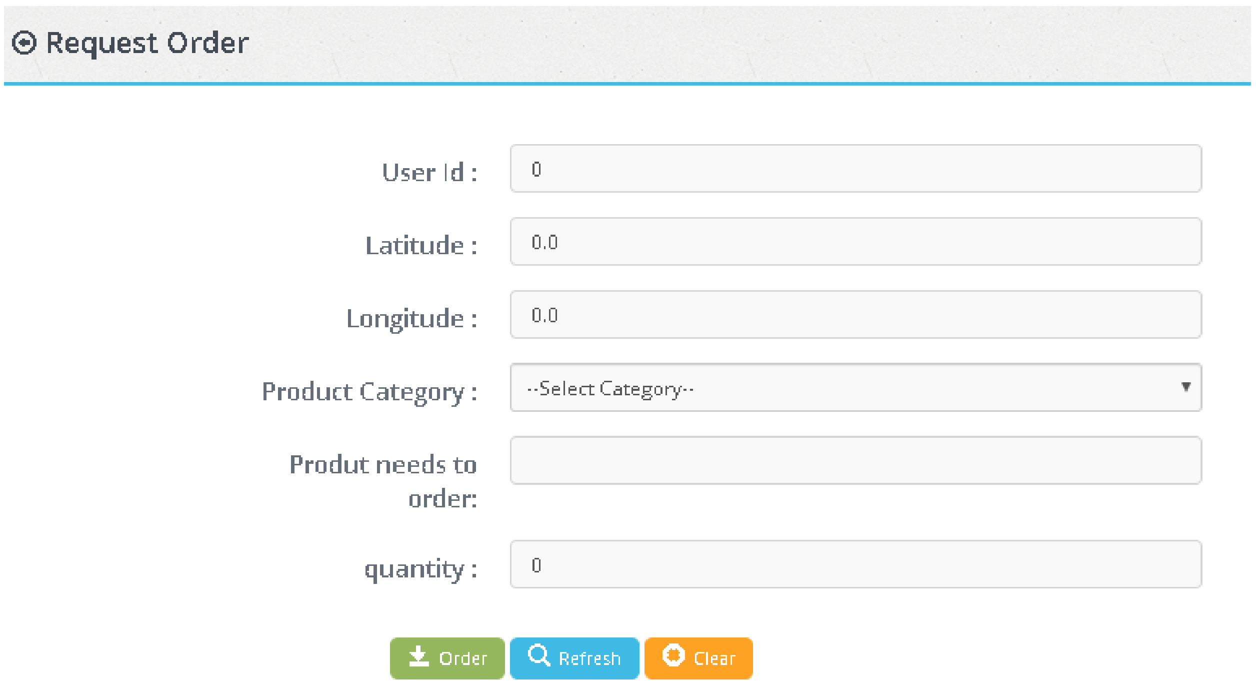
*Figure 6.4: Search agent flow chart diagram*

## 6.3 Database Implementation

There are set of new tables need to create for developing this system and “CustomerOrder” table is the main table of this module. And also there are set of other tables integrate with this table. This can integrate with an existing payment platform by creating these new tables. The SQL scripts in Appendix 04 use for creating those tables.

## 6.4 UI Implementation

User Interfaces is developed as JSP which support for Spring MVC architecture. The following figure 6.5 displays the main screens of this system and customers have to use this UI face to initiate their orders.



| | |
|---|---|
| User Id : | 0 |
| Latitude : | 0.0 |
| Longitude : | 0.0 |
| Product Category : | --Select Category-- |
| Produt needs to order: | |
| quantity : | 0 |

*Figure 6.5: Customer order main UI*

## 6.5 Technology

This system was developed mainly using JAVA and Spring MVC framework used for developing the functionality module. MySQL database is used as the RDMS and data store in this database. Jboss 6.2 server is used as the application server and this can deploy any Java support application server which runs in JDK 1.6. The core module is developed in JDK 1.8 and the APIs have developed in Spring Boot framework which runs in Tomcat server.

## 6.6 Summary

This chapter provided detail description of the implementation of the proposed solution and it described the pseudo codes of each agent type use in this solution with the flow chart diagrams. And also it briefly explained about the DB level implementation and the technology used for implementing this system. Appendix 01 contains the java codes used for implementing this agent class. Next chapter presents the evaluation criteria of this proposed system.

Chapter 7

# Evaluation

## 7.1 Introduction

The previous chapter described the implementation of this novel approach and this chapter presents the Evaluation of this system. It contains several test scenarios and test cases to evaluate the developed system. This is one of the most critical sections in any project and we also considered this as an important part of MASCB development process. This chapter has been structured with three main sections, namely, Experimental Design, Evaluation Strategy and Experimental Results.

## 7.2 Experimental Design

It's very important to follow a proper mechanism to evaluate the system to keep the quality and deliver an error free solution to the end users. So experiments were designed to evaluate the accuracy of the system while developing the system.

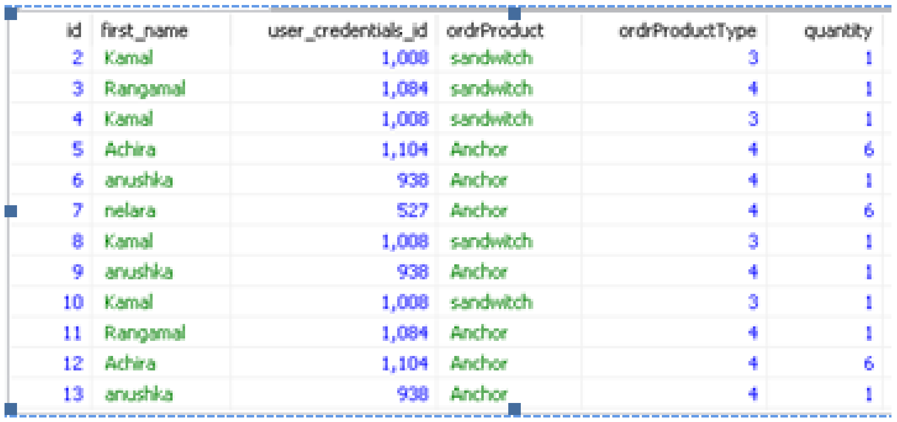
As MASCB is a new system module development it was hard to compare it with an existing this kind of a module. So set of test scenarios and test cases were conducted as explained in the below sections. Unit testing was done while developing the system and integration & UI testing was conducted at the end of the development process

## 7.3. Evaluation Strategy

MASCB start it process when data entering into the system. These data comes into the server and store in the MySQL database. Initially it stores data to the "customerOrder" table with pending status. When the ordering process go forward while communication and negotiation is going on those data also store into the database while modifying the original status of the order request record. And also application server level logs with proper information pass to them displays the current situation at any time and this can be use to tackle any doubtful  incident in MASCB operation process.

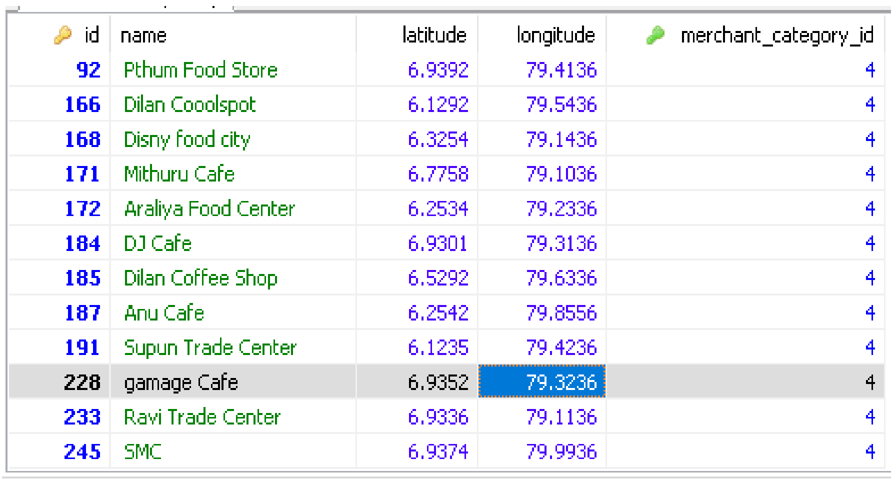
## 7.4 Experimental Results

Number of testing cycles was conducted for evaluating this solution. The following test scenarios are explaining the success of the solution. Different customers will get different prices for the same product based on the conditions like quantity, number of times earlier visited, loyalty points and etc. The following scenario is for ordering a same product from merchants.



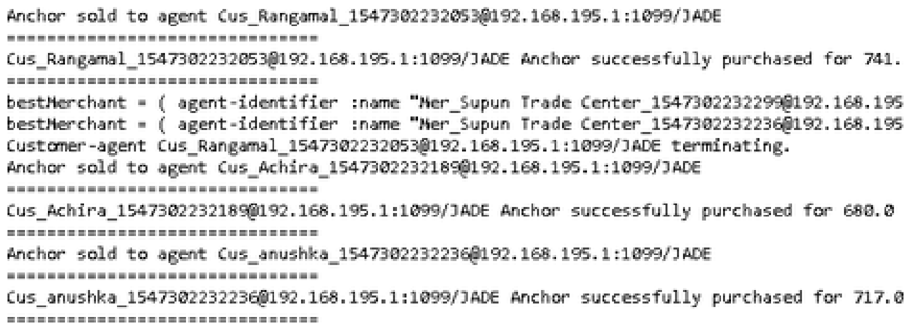
| id | first_name | user_credentials_id | ordrProduct | ordrProductType | quantity |
|---|---|---|---|---|---|
| 2 | Kamal | 1,008 | sandwitch | 3 | 1 |
| 3 | Rangamal | 1,084 | sandwitch | 4 | 1 |
| 4 | Kamal | 1,008 | sandwitch | 3 | 1 |
| 5 | Achira | 1,104 | Anchor | 4 | 6 |
| 6 | anushka | 938 | Anchor | 4 | 1 |
| 7 | nelara | 527 | Anchor | 4 | 6 |
| 8 | Kamal | 1,008 | sandwitch | 3 | 1 |
| 9 | anushka | 938 | Anchor | 4 | 1 |
| 10 | Kamal | 1,008 | sandwitch | 3 | 1 |
| 11 | Rangamal | 1,084 | Anchor | 4 | 1 |
| 12 | Achira | 1,104 | Anchor | 4 | 6 |
| 13 | anushka | 938 | Anchor | 4 | 1 |

*Figure 7.1: Customer order details*



| id | name | latitude | longitude | merchant_category_id |
|---|---|---|---|---|
| 92 | Pthum Food Store | 6.9392 | 79.4136 | 4 |
| 166 | Dilan Cooolspot | 6.1292 | 79.5436 | 4 |
| 168 | Disny food city | 6.3254 | 79.1436 | 4 |
| 171 | Mithuru Cafe | 6.7758 | 79.1036 | 4 |
| 172 | Araliya Food Center | 6.2534 | 79.2336 | 4 |
| 184 | DJ Cafe | 6.9301 | 79.3136 | 4 |
| 185 | Dilan Coffee Shop | 6.5292 | 79.6336 | 4 |
| 187 | Anu Cafe | 6.2542 | 79.8556 | 4 |
| 191 | Supun Trade Center | 6.1235 | 79.4236 | 4 |
| 228 | gamage Cafe | 6.9352 | 79.3236 | 4 |
| 233 | Ravi Trade Center | 6.9336 | 79.1136 | 4 |
| 245 | SMC | 6.9374 | 79.9936 | 4 |

*Figure 7.2: Relevant Merchants*

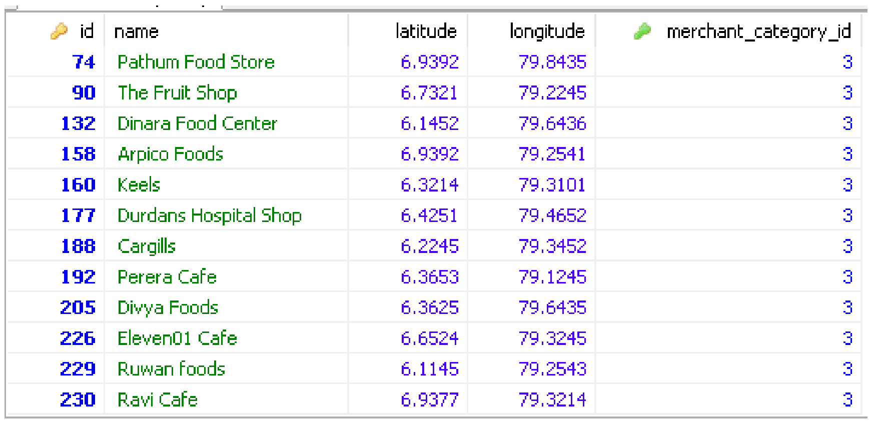


```
Anchor sold to agent Cus_Rangamal_1547302232053@192.168.195.1:1099/JADE
==============================
Cus_Rangamal_1547302232053@192.168.195.1:1099/JADE Anchor successfully purchased for 741.
==============================
bestMerchant = ( agent-identifier :name "Mer_Supun Trade Center_1547302232299@192.168.195
bestMerchant = ( agent-identifier :name "Mer_Supun Trade Center_1547302232236@192.168.195
Customer-agent Cus_Rangamal_1547302232053@192.168.195.1:1099/JADE terminating.
Anchor sold to agent Cus_Achira_1547302232189@192.168.195.1:1099/JADE
==============================
Cus_Achira_1547302232189@192.168.195.1:1099/JADE Anchor successfully purchased for 680.0
==============================
Anchor sold to agent Cus_anushka_1547302232236@192.168.195.1:1099/JADE
==============================
Cus_anushka_1547302232236@192.168.195.1:1099/JADE Anchor successfully purchased for 717.0
==============================
```

*Figure 7.3: Results*

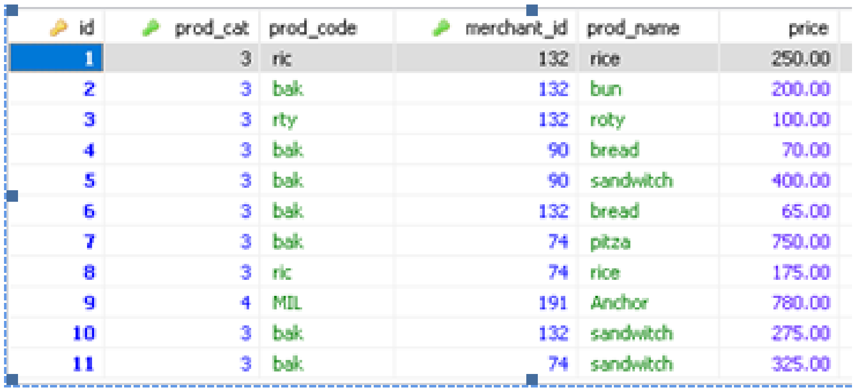
Even though there are buying the same product a merchant in the platform offering the same for different prices as showing in the above figure 7.3.

The Following scenario is for ordering a Sandwich from the merchants in the platform and the search agent propose set suitable merchants based on the selection conditions like distance, customer ratings and etc. These selected merchant agents propose different prices and the system select the merchant with the best offer as showing in the below figures.



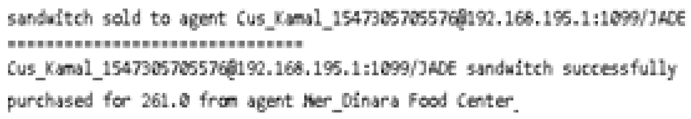
| id | name | latitude | longitude | merchant_category_id |
|---|---|---|---|---|
| 74 | Pathum Food Store | 6.9392 | 79.8435 | 3 |
| 90 | The Fruit Shop | 6.7321 | 79.2245 | 3 |
| 132 | Dinara Food Center | 6.1452 | 79.6436 | 3 |
| 158 | Arpico Foods | 6.9392 | 79.2541 | 3 |
| 160 | Keels | 6.3214 | 79.3101 | 3 |
| 177 | Durdans Hospital Shop | 6.4251 | 79.4652 | 3 |
| 188 | Cargills | 6.2245 | 79.3452 | 3 |
| 192 | Perera Cafe | 6.3653 | 79.1245 | 3 |
| 205 | Divya Foods | 6.3625 | 79.6435 | 3 |
| 226 | Eleven01 Cafe | 6.6524 | 79.3245 | 3 |
| 229 | Ruwan foods | 6.1145 | 79.2543 | 3 |
| 230 | Ravi Cafe | 6.9377 | 79.3214 | 3 |

*Figure 7.4: Related Merchants*



| id | prod_cat | prod_code | merchant_id | prod_name | price |
|---|---|---|---|---|---|
| 1 | 3 | ric | 132 | rice | 250.00 |
| 2 | 3 | bak | 132 | bun | 200.00 |
| 3 | 3 | rty | 132 | roty | 100.00 |
| 4 | 3 | bak | 90 | bread | 70.00 |
| 5 | 3 | bak | 90 | sandwitch | 400.00 |
| 6 | 3 | bak | 132 | bread | 65.00 |
| 7 | 3 | bak | 74 | pitza | 750.00 |
| 8 | 3 | ric | 74 | rice | 175.00 |
| 9 | 4 | MIL | 191 | Anchor | 780.00 |
| 10 | 3 | bak | 132 | sandwitch | 275.00 |
| 11 | 3 | bak | 74 | sandwitch | 325.00 |

*Figure 7.5: Merchant Inventory Details*



```
sandwitch sold to agent Cus_Kamal_1547305705576@192.168.195.1:1099/JADE
===============================
Cus_Kamal_1547305705576@192.168.195.1:1099/JADE sandwitch successfully
purchased for 261.0 from agent Mer_Dinara Food Center_
```

*Figure 7.6: Results*

Like in earlier scenarios this proposed solution helps the customers to order product online from the merchants in the platform in an easier manner. While developing the system incremental evaluation is done but we can do some advanced evaluation once this apply in a production environment by considering follows.

- Retention of customers after some time.
- Merchant revenue incensement
- Customer satisfaction
- Merchant satisfaction

## 7.5 Summary

This Chapter described about the test scenarios and test cases considered for evaluating this system. Testing was carried out while developing this system to achieve a quality product without waiting until the end of this project. It was given more attention for evaluating the system as whole quality of this product depends on the evaluation process. In next chapter we will explain the conclusion and further work of our research project.

Chapter 8

# Conclusion and further work

## 8.1 Introduction

Chapter 7 described the evaluation section of our research project and it was one of the critical sections of this project. This is the final section of this thesis and will conclude the result obtained in previous chapter. And also it will describe the limitations and further improvements of the developed system.

## 8.2 Conclusion

The above explained customer driven bargaining system may make people life easier and may bring more advantages for both customer and merchant. Currently there are lots of merchant controlling bargaining systems in the market but this system introduces a customer driven bargaining system. MAS based solution is ideal for this kind of complex system and it handles the complexity of negotiation, coordination and communication of the dynamic environment. Platform may provide ideal solutions and recommendations based on the past data system keeps and the developed system contains APIs, which can be used to connect with any existing mobile and e-commerce App to bring this feature as a value added service to their customers. This can inevitably lead to greater levels of customer satisfaction and help the merchants achieve a competitive edge over other competitors.

When it comes to the objectives identified in this study, Critical review of digital payment platform challenges has done and identifies the challenges of this kind of platforms and it has included in the second chapter of this thesis. In depth study of technology used for order products in payment platforms also thoroughly investigate during this project as explained in above chapter two. Critically analyzed the applicability of Genetic algorithms when finding quick optimal solutions in dynamic environment as mentioned in chapter three and system was thoroughly evaluated and the test scenarios and test cases also included in chapter seven.

## 8.3 Limitations of the System

The developed MASCB system is a web based system and this should be provided through a mobile application and it may bring more convenience to the both customers and the merchants to use the application in a simpler manner. Customer location should automatically get through the browser or from the mobile app and currently users need to input their location to the system and this may be a hassle for the customers. These are the identified limitations of this project and this may overcome with future works to deliver a better service to the end users.

## 8.4 Future Work

MASCB can be integrated with any third part delivery platform through the APIs and this has opened a new business opportunity as well. So this module can be expanded while increasing the scope and it will be more helpful for both customers and merchants in an echo system. And also to overcome the above mentioned limitations it’s need to developed a mobile application or needs to integrate with an existing mobile application to offer more user experience to the end users.

## 8.5 Summary

The first chapter of this thesis provided a detailed introduction about this project, by highlighting its aims and objectives, the background, the identified problem, the proposed solution and the resource requirements. In chapter two it was discussed about the existing digital payment platforms, their limitations, MAS in industrial landscape and the problem identification. Subsequently, the identified technologies were discussed in greater detail in the third chapter. The fourth chapter provided details about the novel Approach to MASCB by highlighting its hypothesis, inputs, outputs, process and features. The fifth and the sixth chapters of the thesis explained about the design and the implementation of this system respectively. It has mentioned how it was evaluated in chapter seven. Finally, in this chapter, the conclusions for this developed system were discussed by indicating the limitations and future works.

# Reference

[1] T. V. S. O.A. Sosnovskiy, “Multi-Agent Systems for E-Commerce.” International Conference on Pattern Recognition and Information Processing (PRIP'2009).

[2] S. K. Zhang Yuheng, “A MULTI AGENT BASED E-SHOPPING SYSTEM,” *Volume 2, No. 4, April 2011 Journal of Global Research in Computer Science*.

[3] S.-U. G. Feng Hua, “A Multi-Agent Architecture for Electronic payment,” *International Journal of Information Technology and Decision Making · November 2011*.

[4] “A Comprehensive Analysis on Multi Agent Decision Making Systems,” *Indian Journal of Science and Technology, Vol 9(11), DOI: 10.17485/ijst/2016/v9i11/89261, March 2016*.

[5] Y. P. T. Finin, “A Multi-Agent System for Enterprise Integration.” International Journal of Agile Manufacturing (December 1, 1998)

[6] M. C. G. Bih-Ru Lea, “A prototype multi-agent ERP system.” (2005) : an integrated architecture and a conceptual framework. Technovation 25(4):433–441

[7] Michal Kopys1 and Wojciech Jedruch Rafal Krolikowski, “Self-Organization in Multi-Agent Systems Based on Examples of Modeling Economic Relationships between Agents.”

[8] Paulo Leitao, “Multi-agent Systems in Industry: Current Trends & Future Challenges.” [Online]. Available: https://pdfs.semanticscholar.org/234c/6a74f071697baf3f8211ba5244e111971850.pdf. [Accessed: 27-Oct-2017].

[9] A. Karunananda and L. Perera, “Using a multi-agent system for supply chain management,” *International Journal of Design & Nature and Ecodynamics*, vol. 11, pp. 107–115, Apr. 2016.

[10] T. Moyaux, B. Chaib-draa, and S. D’Amours, “Supply Chain Management and Multiagent Systems: An Overview,” in *Multiagent based Supply Chain Management,* Springer, Berlin, Heidelberg, 2006, pp. 1–27.

[11] A. Kulasekera, R. Gopura, K. T. M. U. Hemapala, and N. Perera, “A Review on Multi-agent Systems in Microgrid Applications,” in *2011 IEEE PES International Conference on Innovative Smart Grid Technologies-India, ISGT India 2011*, 2011.

[12] “Integrating mobile agent technology with MAS.” [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.467.2880&rep=rep1&type=pdf. [Accessed: 27-Oct-2017].

[13] Andrew Lucas, “The OASIS Air Traffic Management System.” [Online]. Available: https://www.researchgate.net/publication/2702604_The_OASIS_Air_Traffic_Management_System. [Accessed: 28-Oct-2017].

[14] “Multi agent based approach to assist the design process of 3D game environments.” [Online]. Available: https://www.researchgate.net/publication/261449497_Multi_agent_based_approach_to_assist_the_design_process_of_3D_game_environments. [Accessed: 27-Oct-2017].

[15] Dante I. Tapia, Ricardo S. Alonso, Óscar García, Juan M. Corchado, “Wireless_Sensor_Networks_Real-Time_Locat.” (2011) International Symposium on Distributed Computing and Artificial Intelligence. 10.1007/978-3-642-19934-9

[16] Michal Pˇechouˇcek · Vladimír Maˇrík, “Industrial deployment of multi-agent technologies: review and selected case studies.” [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.9080&rep=rep1&type=pdf. [Accessed: 28-Oct-2017].

[17] J.M. Bradshaw, *An Introduction to Software Agents*, Pp. 3-46. Menlo Park: AAAI Press., 1997.

[18] John McCall, “GA for modelling & optimisation,” *Journal of Computational and Applied Mathematics 184 (2005) 205–222*.

[19] Erol Kazan “An investigation of digital payment platform designs: a comparative study of four european solutions” *Copenhagen Business School, Howitzvej 60, 2000 Frederiksberg, Denmark, eka.itm@cbs.dk.*

[20] Kazan, Erol “Towards a Market Entry Framework for Digital Payment Platforms” Copenhagen Business School, Howitzvej 60, 2000 Frederiksberg, Denmark, eka.itm@cbs.dk.*Communications of the Association for Information Systems ISSN: 1529-3181*

[21] Zlatko Bezhovski Goce Delchev “An The Future of the Mobile Payment as Electronic Payment System” , *University, Krste Misirkov No.10-A, Stip, Macedonia. European Journal*

*of Business and Management, ISSN 2222-1905 (Paper) ISSN 2222-2839 (Online), Vol.8, No.8, 2016*

[22] Karbowska-Chilinska, Joanna & Zabielski, Paweł. (2012). A Genetic Algorithm vs. Local Search Methods for Solving the Orienteering Problem in Large Networks. 7828. 10.1007/978-3-642-37343-5_2

# Appendixes

## Appendix 01

### Manager Agent:

```
package lk.ac.mrt.msc.ai.masbe.agent;

import jade.Boot;
import jade.core.Agent;
import jade.core.AID;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.behaviours.*;
import jade.wrapper.ContainerController;
import lk.ac.mrt.msc.ai.masbe.Genetic.MerchantSelectionModule;
import lk.ac.mrt.msc.ai.masbe.Genetic.MerchantSelectionModule.Record;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;

import org.jenetics.util.ISeq;



/**
 * @author udayanga This class is for Manager Agent
 *
 */
public class ManagerAgent extends Agent {

        String param1;

        protected void setup() {

                Object[] args = getArguments();
                if (args != null && args.length > 0) {

                        // Add a TickerBehaviour that schedules a request to service provider agents
                        // every 18 seconds
                        addBehaviour(new TickerBehaviour(this, 18000) {
                                protected void onTick() {
                                        // all agent types initiate here
                                        initiateAgents();
                                }
                        });
                } else {
```

```
				// Make the agent terminate
				System.out.println("No target service specified");
				doDelete();
			}
		}

		// Put agent clean-up operations here
		protected void takeDown() {
			// Printout a dismissal message
			System.out.println("Service Requester-agent " + getAID().getName() + " terminating.");
		}

		public void initiateAgents() {
			try {
				/*
				 *
				 * All agent types are created here
				 */
				jade.core.Runtime rt = jade.core.Runtime.instance();
				Profile p = new ProfileImpl();

				ContainerController agentContainer = rt.createAgentContainer(p);
				/* Get DB Connection */
				String driverName = "com.mysql.jdbc.Driver";
				try {
					Class.forName(driverName);
				} catch (ClassNotFoundException e) {
					e.printStackTrace();
				}

				Connection myconnection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql", "root",
						"root123");

				PreparedStatement queryCust = null;
				PreparedStatement queryMerc = null;
				PreparedStatement queryUpdCust = null;
				PreparedStatement queryUpdMer = null;
				queryCust = myconnection.prepareStatement(
						"select 
first_name,ordrProduct,latitude,longitude,id,ordrProductType,user_credentials_id,quantity from ipaydb.customerOrder where status = 'A' and 
ordrStatus = 'A'");

				queryUpdCust = myconnection
						.prepareStatement("update ipaydb.customerOrder set ordrStatus ='I' where id = ? ");
				// queryUpdMer = myconnection.prepareStatement("update ipaydb.merchant set
				// ordrStatus ='I' where id = ? ");

				ResultSet rs = queryCust.executeQuery();
				int merchantcatId;

				while (rs.next()) { // when there is a row next, do the loop

					merchantcatId = rs.getInt(6);

					agentContainer.createNewAgent("Cus_" + rs.getString(1) + "_" + System.currentTimeMillis(),
							"lk.ac.mrt.msc.ai.masbe.agent.CustomerAgent", new Object[] { 
rs.getString(2), rs.getString(3),
										rs.getString(4), rs.getString(7), rs.getString(8), 
rs.getString(5) })
							.start();

					queryUpdCust.setString(1, rs.getString(5));
					queryUpdCust.executeUpdate();

					queryMerc = myconnection.prepareStatement(
							"select name,latitude,longitude,merchant_category_id,id,DiscntStatus from 
ipaydb.merchant where status = 'A' and ordrStatus = 'A' and merchant_category_id =?");
					queryMerc.setLong(1, merchantcatId);
```

```
                                        ResultSet rsS = queryMerc.executeQuery();

                                        /**
                                         * call Genetic Algorithm process for search merchants
                                         */
                                        int noOfMerc = 3;
                                        int mrCount = 0;
                                        Record record = null;
                                        ArrayList<Record> RecordList = new ArrayList<Record>();
                                        ArrayList<MerchantSelectionModule.ResultRecord> ResultList = new
ArrayList<MerchantSelectionModule.ResultRecord>();
                                        while (rsS.next()) {
                                                record = new Record(Integer.parseInt(rsS.getString(5)), 3, rsS.getString(1),
                                                                Double.parseDouble("3.00"), Double.parseDouble("4.00"),
                                                                Double.parseDouble(rsS.getString(2)),
Double.parseDouble(rsS.getString(3)),
                                                                rsS.getString(6));
                                                RecordList.add(record);
                                                mrCount++;
                                        }
                                        noOfMerc = (mrCount < noOfMerc) ? mrCount : noOfMerc;

                                        MerchantSelectionModule merchantSelectionModule = new
MerchantSelectionModule(ISeq.of(RecordList),
                                                        noOfMerc);

                                        ResultList = merchantSelectionModule.selectionProcess(RecordList, noOfMerc);
                                        for (MerchantSelectionModule.ResultRecord reslt : ResultList) {
                                                System.out.println(" name :" + reslt.nameOfmerchant);
                                                System.out.println(" id :" + reslt.merchantId);
                                                int count = reslt.merchantId.length() - reslt.merchantId.replace(",", "").length();
                                                String[] mrId = reslt.merchantId.split(",", count + 1);
                                                String[] mrName = reslt.nameOfmerchant.split(",", count + 1);
                                                String[] mrLatitude = reslt.latitude.split(",", count + 1);
                                                String[] mrLongitude = reslt.longitude.split(",", count + 1);
                                                String[] mrDisCntSts = reslt.discntStatus.split(",", count + 1);
                                                int x = 0;
                                                while (x <= count) {
                                                        System.out.println("merchant creation = " + x);
                                                        String merchantId = mrId[x];
                                                        String merchantName = mrName[x];
                                                        String merchantLatd = mrLatitude[x];
                                                        String merchantLontd = mrLongitude[x];
                                                        String merchantds = mrDisCntSts[x];
                                                        System.out.println("merchantName " + merchantName);
                                                        System.out.println("merchantLatd " + merchantLatd);
                                                        System.out.println("Long.parseLong(merchantId) " +
Long.parseLong(merchantId));
                                                        System.out.println("merchantId " + merchantId);
                                                        System.out.println("merchantds " + merchantds);

                                                        agentContainer.createNewAgent("Mer_" + merchantName + "_" +
System.currentTimeMillis(),
                                                                        "lk.ac.mrt.msc.ai.masbe.agent.MerchantAgent",
                                                                        new Object[] { merchantName, merchantLatd,
merchantLontd,
                                                                                        Integer.toString(merchantcatId),
Long.parseLong(merchantId), merchantId,
                                                                                        merchantds })
                                                                        .start();

                                                        x++;
                                                }

                                        }

                                        ;
```

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

		}
}
```

## Customer Agent:

```
package lk.ac.mrt.msc.ai.masbe.agent;

import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Hashtable;

import javax.management.loading.PrivateClassLoader;

import jade.core.AID;
import jade.core.Agent;
import jade.core.AgentContainer;
import jade.core.MainContainer;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.StringACLCodec;

/**
 * @author udayanga
 * This class is for Customer Agent
 */
public class CustomerAgent extends Agent {

	private Hashtable inventory;
	// properties of customer agent
	private String expectProduct;
	private double expectPrice;
	private String latitude;
	private String longitude;
	private int userId;
	private int quantity;
	private int orderId;
	PreparedStatement queryUpdOdr = null;

	// private MerchantAgent currentMerchant;

	// list of search agents
	private AID[] searchAgents;

	// list of merchants
	private AID[] merchantAgents;

	// Put agent initializations here
	protected void setup() {
		System.out.println("Customer-agent " + getAID().getName() + " is ready.");
		inventory = new Hashtable();
		Object[] paramRequestProduct = getArguments();
		if (paramRequestProduct != null && paramRequestProduct.length > 0) {
			expectProduct = (String) paramRequestProduct[0];
			latitude = (String) paramRequestProduct[1];
			longitude = (String) paramRequestProduct[2];
			userId =  Integer.parseInt((String) paramRequestProduct[3]) ;
			quantity =  Integer.parseInt((String) paramRequestProduct[4]) ;
			orderId =  Integer.parseInt((String) paramRequestProduct[5]) ;
			System.out.println("Expect product is " + expectProduct + " Customer location latitude is "+ latitude + " 
longitude is " + longitude);


			addBehaviour(new TickerBehaviour(this, 30000) {
				protected void onTick() {
					DFAgentDescription template = new DFAgentDescription();
					ServiceDescription sd = new ServiceDescription();
					sd.setType("search");
					template.addServices(sd);
					try {
						// Directory Facilitator DFService
						DFAgentDescription[] result = DFService.search(myAgent, template);
						searchAgents = new AID[result.length];
						for (int i = 0; i < result.length; ++i) {
							searchAgents[i] = result[i].getName();
						}
					} catch (FIPAException fe) {
						fe.printStackTrace();
					}

					// Perform the request
					myAgent.addBehaviour(new RequestMerchant());
				}
			});
		} else {
			// Make the agent terminate
			System.out.println("No expect product specified");
			doDelete();
		}
	}

	// Put agent clean-up operations here
	protected void takeDown() {
		// Printout a dismissal message
		System.out.println("Customer-agent " + getAID().getName() + " terminating.");
	}

	public void updateCatalogue(final String title, final int price) {
		addBehaviour(new OneShotBehaviour() {
			public void action() {
				inventory.put(title, new Integer(price));
			}
		});
	}

	/**
	 * Inner class bestSeller. This is the behaviour used by customer agents to
	 * request product from merchant agents
	 */
	private class RequestMerchant extends Behaviour {
		private AID bestMerchant; // The agent who provides the best offer
		private double bestPrice; // The best offered price
		private int repliesCnt = 0; // The counter of replies from merchant agents
		private String BMLatitude = null;
		private String BMLongitude = null;
		private MessageTemplate mt; // The template to receive replies
```

```
                private int step = 0;
                private String merLocation;
                private MerchantAgent currMerchant;
                // private AgentContainer localContainer = AgentContainer.MAIN_CONTAINER_NAME;
                private String localContainer = AgentContainer.MAIN_CONTAINER_NAME;
                private AgentContainer localContainer1;

                public void action() {



                        switch (step) {
                        case 0:
                                // get reply from search agent

                                ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
                                for (int i = 0; i < searchAgents.length; ++i) {
                                        cfp.addReceiver(searchAgents[i]);
                                }
                                cfp.setContent(expectProduct);// +"$"+latitude+"$"+longitude
                                cfp.setConversationId("food-trade");
                                cfp.setReplyWith("cfp" + System.currentTimeMillis()); // Unique value
                                myAgent.send(cfp);
                                // Prepare the template to get proposals
                                mt = MessageTemplate.and(MessageTemplate.MatchConversationId("food-trade"),
                                                MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
                                step = 1;

                                break;
                        case 1:

                                ACLMessage replyFromSearchAgent = myAgent.receive(mt);
                                if (replyFromSearchAgent != null) {

                                        if (replyFromSearchAgent.getPerformative() == ACLMessage.PROPOSE) {
                                                String searchAgentMessage = replyFromSearchAgent.getContent();

                                                String sub = "@@";
                                                String temp = searchAgentMessage.replace(sub, "");
                                                int occ = (searchAgentMessage.length() - temp.length()) / sub.length();


                                                try {
                                                        merchantAgents = new AID[occ + 1];
                                                        String[] merchantAgentStr = searchAgentMessage.split("@@",
occ + 1);

                                                        for (int i = 0; i < occ + 1; ++i) {

                                                                merchantAgents[i] = new ~~AID~~(merchantAgentStr[i]);
                                                        }

                                                        ACLMessage cfpMerchants = new
ACLMessage(ACLMessage.CFP);
                                                        for (int i = 0; i < merchantAgents.length; ++i) {
                                                                cfpMerchants.addReceiver(merchantAgents[i]);
                                                        }

        cfpMerchants.setContent(expectProduct+"@"+userId+"@"+quantity);// +"$"+latitude+"$"+longitude
                                                        cfpMerchants.setConversationId("food-trade");
                                                        cfpMerchants.setReplyWith("cfp" + System.currentTimeMillis());
// Unique value
                                                        myAgent.send(cfpMerchants);

                                                        // Prepare the template to get proposals
                                                        mt =
MessageTemplate.and(MessageTemplate.MatchConversationId("food-trade"),
        MessageTemplate.MatchInReplyTo(cfpMerchants.getReplyWith()));
                                                } catch (Exception ex) {
                                                        ex.printStackTrace();
                                                }

                                                step = 2;
                                        } else {
                                                System.out.println("There is no any search agent in the platform ");
                                                step = 5;
                                        }

                                } else {
                                        block();
                                }

                                break;

                        case 2:

                                // Receive all proposals/refusals from merchant agents
                                ACLMessage reply = myAgent.receive(mt);
                                if (reply != null) {
                                        // Reply received
                                        if (reply.getPerformative() == ACLMessage.PROPOSE) {
                                                // split merchant message to get parameters
                                                String realMessage = reply.getContent();

                                                String[] arrOfStr = realMessage.split("@", 3);

                                                String Crprice = arrOfStr[0];

                                                BMLatitude = arrOfStr[1];
        //                                      System.out.println("BMLatitude " + BMLatitude);
                                                BMLongitude = arrOfStr[2];
        //                                      System.out.println("BMLongitude " + BMLongitude);
                                                // This is an offer
                                                double price = Double.valueOf(Crprice);
                                                // int price = Integer.parseInt(reply.getContent());
                                                if (bestMerchant == null || price < bestPrice) {
                                                        // This is the best offer at present
                                                        bestPrice = price;
                                                        bestMerchant = reply.getSender();
                                                        System.out.println("bestMerchant = " + bestMerchant);

                                                }
                                        }
                                        repliesCnt++;
                                        if (repliesCnt >= merchantAgents.length) {
                                                // We received all replies
                                                step = 3;
                                        }
                                } else {
                                        block();
                                }

                                break;
                        case 3:

                                // Send the purchase order to the merchant that provided the best offer
                                ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                                order.addReceiver(bestMerchant);
                                order.setContent(expectProduct+"@"+ quantity); // +"$"+BMLatitude+"$"+BMLongitude
                                order.setConversationId("food-trade");
                                order.setReplyWith("order" + System.currentTimeMillis());
                                myAgent.send(order);
                                // Prepare the template to get the purchase order reply
```

```
                                        mt = MessageTemplate.and(MessageTemplate.MatchConversationId("food-trade"),
                                                        MessageTemplate.MatchInReplyTo(order.getReplyWith()));
                                        step = 4;

                                        break;
                                case 4:
                                        // Receive the purchase order reply

                                        reply = myAgent.receive(mt);
                                        if (reply != null) {
                                                // Purchase order reply received
                                                if (reply.getPerformative() == ACLMessage.INFORM) {
                                                        String messageContent = reply.getContent();
                                                        // Purchase successful. We can terminate
                                                        System.out.println("=============================="+ "\n"
+getAID().getName()+ " " +expectProduct + " successfully purchased for " + bestPrice + " from agent "
                                                                        + reply.getSender().getName() +"Price = "  + "\n" +
"==============================");

                                                        String driverName = "com.mysql.jdbc.Driver";
                                                        try {
                                                                Class.forName(driverName);

                                                        Connection myconnection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql", "root",
                                                                        "root123");
                                                        queryUpdOdr = myconnection
                                                        .prepareStatement("update ipaydb.customerOrder set buyMerchantId =?
where id = ? ");

                                                        queryUpdOdr.setLong(1, Integer.parseInt(messageContent));
                                                        queryUpdOdr.setLong(2, orderId);
                                                        queryUpdOdr.executeUpdate();
                                                        } catch (ClassNotFoundException e) {
                                                                e.printStackTrace();
                                                        }catch (Exception e) {
                                                                e.printStackTrace();
                                                        }

                                                        myAgent.doDelete();
                                                } else {
                                                        System.out.println("Attempt failed: requested " + expectProduct + "  already
sold.");
                                                }

                                                step = 5;
                                        } else {
                                                block();
                                        }

                                        break;
                                }
                        }

                        public boolean done() {
                                if (step == 3 && bestMerchant == null) {
                                        System.out.println("Attempt failed: " + expectProduct + " not available for sale");
                                }
                                return ((step == 3 && bestMerchant == null) || step == 5);
                        }
                } // End of inner class RequestMerchant

}
```

## Merchant Agent:

```
package lk.ac.mrt.msc.ai.masbe.agent;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Hashtable;

import jade.core.Agent;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import lk.ac.mrt.msc.ai.masbe.view.MerchantGui;

/**
 * @author udayanga
 * This class is for Merchant Agent
 */
public class MerchantAgent extends Agent {

					// price list include in the inventory
					private Hashtable inventory;
					// The GUI by means of which the user can add items to inventory
					private MerchantGui myGui;
					// merchant location
					private String merLatitude;
					private int merchantId;

					public String getMerLatitude() {
						return merLatitude;
					}

					public void setMerLatitude(String merLatitude) {
						this.merLatitude = merLatitude;
					}

					private String merLongitude;

					// Put agent initializations here
					protected void setup() {
						// Create the catalogue
						inventory = new Hashtable();
						Object[] paramMerLocation = getArguments();
						try {
							Profile p = new ProfileImpl();
							String driverName = "com.mysql.jdbc.Driver";
							Connection myconnection = 
DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql", "root",
									"root123");
							PreparedStatement queryMerProds = null;
							PreparedStatement queryUpdMerProds = null;

							queryMerProds = myconnection.prepareStatement(
									"select 
id,merchant_id,prod_cat,prod_name,price,quantity,DiscntRate from ipaydb.merchantinventory  where status ='A'  and merchant_id = ?");
							queryMerProds.setString(1, (String) paramMerLocation[5]);
							merchantId = Integer.parseInt((String) paramMerLocation[5]);
							ResultSet rsS = queryMerProds.executeQuery();
							while (rsS.next()) {
```

```
						updateCatalogue(rsS.getString(4), rsS.getInt(5));
				}

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

		// Create and show the GUI
		// myGui = new MerchantGui(this);
		// myGui.showGui();
		// updateCatalogue(title, Integer.parseInt(price));

		/*
		 * if (((String) paramMerLocation[0]).equals("Dinara Food Center") ) {
		 * updateCatalogue("rice", 100); System.out.println(" 0 inex "+(String)
		 * paramMerLocation[0]); }else { System.out.println(" 0 inex else "+(String)
		 * paramMerLocation[0]); updateCatalogue("bread", 300); }
		 */

		this.merLatitude = (String) paramMerLocation[1];

		this.merLongitude = (String) paramMerLocation[2];

		// Register the book-selling service in the yellow pages
		DFAgentDescription dfd = new DFAgentDescription();
		dfd.setName(getAID());
		ServiceDescription sd = new ServiceDescription();
		sd.setType("food-selling");
		sd.setName("JADE-food-trading");
		dfd.addServices(sd);
		try {
				DFService.register(this, dfd);
		} catch (FIPAException fe) {
				fe.printStackTrace();
		}

		// Add the behaviour serving queries from buyer agents
		addBehaviour(new OfferRequestsServer());

		// Add the behaviour serving purchase orders from buyer agents
		addBehaviour(new PurchaseOrdersServer());
}

public String getMerLongitude() {
		return merLongitude;
}

public void setMerLongitude(String merLongitude) {
		this.merLongitude = merLongitude;
}

// Put agent clean-up operations here
protected void takeDown() {
		// Deregister from the yellow pages
		try {
				DFService.deregister(this);
		} catch (FIPAException fe) {
				fe.printStackTrace();
		}
		// Close the GUI
		myGui.dispose();
		// Printout a dismissal message
		System.out.println("merchant-agent " + getAID().getName() + " terminating.");
}

/**
 * This is invoked by the GUI when the user adds a new book for sale
```

```
                                         */
                                        public void updateCatalogue(final String title, final int price) {
                                              addBehaviour(new OneShotBehaviour() {
                                                      public void action() {
                                                              inventory.put(title, new Integer(price));
                                                              System.out.println(title + " inserted into inventory. Price = " + price);
                                                      }
                                              });
                                        }

                                        /**
                                         * Inner class OfferRequestsServer. This is the behaviour used by Book-seller
                                         * agents to serve incoming requests for offer from buyer agents. If the
                                         * requested book is in the local catalogue the seller agent replies with a
                                         * PROPOSE message specifying the price. Otherwise a REFUSE message is sent
                                         * back.
                                         */
                                        private class OfferRequestsServer extends CyclicBehaviour {
                                              public void action() {

                                                      MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
                                                      ACLMessage msg = myAgent.receive(mt);
                                                      int custpoints = 0;
                                                      int custOrderCnt = 0;
                                                      double sellingPrice = 0;
                                                      if (msg != null) {
                                                              // CFP Message received. Process it
                                                              String title = msg.getContent();
                                                              ACLMessage reply = msg.createReply();

                                                              String[] arrOfStr = title.split("@", 3);

                                                              title = arrOfStr[0];
                                                              int userId = Integer.parseInt(arrOfStr[1]);
                                                              int quantity = Integer.parseInt(arrOfStr[2]);

                                                              Integer price = (Integer) inventory.get(title);
                                                              if (price != null) {
                                                                      // The requested item is available for sale. Reply with the price
                                                                      reply.setPerformative(ACLMessage.PROPOSE);
                                                                      reply.setContent(String.valueOf(price.intValue()) + "@" +
merLatitude + "@" + merLongitude);
                                                                      // get discounted price

                                                                      try {
                                                                              Profile p = new ProfileImpl();
                                                                              String driverName = "com.mysql.jdbc.Driver";
                                                                              Connection myconnection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql",
                                                                                              "root", "root123");

                                                                              PreparedStatement querySelectDiscnts = null;
                                                                              querySelectDiscnts = myconnection.prepareStatement(
                                                                                              "select merchant_id,points from
ipaydb.merchantcustloyality where cust_id =? and merchant_id = ? and status= 'A' ");
                                                                              querySelectDiscnts.setLong(1, userId);
                                                                              querySelectDiscnts.setLong(2, merchantId);

                                                                              ResultSet rs = querySelectDiscnts.executeQuery();

                                                                              while (rs.next()) {

                                                                                      custpoints =
Integer.parseInt(rs.getString(1));

                                                                              }

                                                                              // check whether a new customer
```

```
								querySelectDiscnts = myconnection.prepareStatement(
										"select count(id) from
ipaydb.customerOrder where user_credentials_id = ?");
								querySelectDiscnts.setLong(1, userId);
								rs = querySelectDiscnts.executeQuery();
								while (rs.next()) {

									custOrderCnt =
Integer.parseInt(rs.getString(1));

								}

								if (custpoints > 0) {
									if (quantity >= 5) {
										sellingPrice =
Math.round(price.intValue() * .95) - 25;

									} else {
										sellingPrice = price.intValue() -
25;

									}
								} else {

									sellingPrice = price.intValue();
								}

								if (custOrderCnt == 1) {
									sellingPrice = Math.round(sellingPrice *
.75);
								} else if(custOrderCnt >= 5) {
									sellingPrice = Math.round(sellingPrice *
.95);

								}
								reply.setContent(String.valueOf(sellingPrice) + "@" +
merLatitude + "@" + merLongitude);

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

						} else {
							// The requested book is NOT available for sale.
							reply.setPerformative(ACLMessage.REFUSE);
							reply.setContent("not-available");
						}
						myAgent.send(reply);
					} else {
						block();
					}
				}
			} // End of inner class OfferRequestsServer

			/**
			 * Inner class PurchaseOrdersServer. This is the behaviour used by Book-seller
			 * agents to serve incoming offer acceptances (i.e. purchase orders) from buyer
			 * agents. The seller agent removes the purchased book from its catalogue and
			 * replies with an INFORM message to notify the buyer that the purchase has been
			 * sucesfully completed.
			 */
			private class PurchaseOrdersServer extends CyclicBehaviour {
				public void action() {

					try {
						Profile p = new ProfileImpl();
						String driverName = "com.mysql.jdbc.Driver";
					Connection myconnection = 
DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql", "root",
							"root123");

					PreparedStatement queryUpdMerProds = null;

					MessageTemplate mt = 
MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
					ACLMessage msg = myAgent.receive(mt);
					if (msg != null) {
						// ACCEPT_PROPOSAL Message received. Process it
						String title = msg.getContent();
						String[] arrOfStr = title.split("@", 3);
						String crProduct = arrOfStr[0];
						title = crProduct;
						String quantity = arrOfStr[1];
						title = crProduct;
						ACLMessage reply = msg.createReply();
						Integer price = (Integer) inventory.remove(title);

						if (price != null) {
							queryUpdMerProds = myconnection.prepareStatement(
									"update 
ipaydb.merchantinventory set quantity = quantity-? where prod_name = ? and price = ?");
							queryUpdMerProds.setString(1, quantity);
							queryUpdMerProds.setString(2, title);
							queryUpdMerProds.setInt(3, price);
							queryUpdMerProds.executeUpdate();

							reply.setPerformative(ACLMessage.INFORM);
							reply.setContent(String.valueOf(merchantId));

							System.out.println(title + " sold to agent " + 
msg.getSender().getName());
						} else {
							// The requested book has been sold to another buyer in 
the meanwhile .
							reply.setPerformative(ACLMessage.FAILURE);
							reply.setContent("not-available");
						}
						myAgent.send(reply);
					} else {
						block();
					}
				} catch (Exception ex) {
					ex.printStackTrace();
				}
			}
		} // End of inner class OfferRequestsServer

}
```

## Search Agent:

```
package lk.ac.mrt.msc.ai.masbe.agent;

import java.util.ArrayList;
import java.util.List;

////Directory Facilitator  DFService
import javax.management.loading.PrivateClassLoader;

import jade.core.AID;
import jade.core.Agent;
import jade.core.AgentContainer;
import jade.core.MainContainer;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

/**
 * @author udayanga This is for Search Agent
 *
 */
public class SearchAgent extends Agent {
				// properties of customer agent
				private String expectProduct;
				private double expectPrice;
				private String latitude;
				private String longitude;
				private AID custAgent;
				int step;

				MessageTemplate mtMerchantsRep = MessageTemplate.MatchPerformative(ACLMessage.CFP);

				// list of merchants
				private AID[] merchantAgents;
				private AID[] merchantAgentsAcc;

				// Put agent initializations here
				protected void setup() {
					// Register the merchant service in the yellow pages

					DFAgentDescription dfd = new DFAgentDescription();
					dfd.setName(getAID());
					ServiceDescription sd = new ServiceDescription();
					sd.setType("search");
					sd.setName("MAS-trading");
					dfd.addServices(sd);
					try {
						DFService.register(this, dfd);
					} catch (FIPAException fe) {
						fe.printStackTrace();
					}

					addBehaviour(new SuggestMerchants());

				}

				private class SuggestMerchants extends CyclicBehaviour {
					public void action() {

						MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
						ACLMessage msg = myAgent.receive(mt);
						if (msg != null) {

							String title = msg.getContent();
							expectProduct = title;
							custAgent = msg.getSender();
							ACLMessage reply = msg.createReply();
							if (title != null) {
								DFAgentDescription template = new DFAgentDescription();
								ServiceDescription sd = new ServiceDescription();
								sd.setType("food-selling");
								template.addServices(sd);
								try {
```

```
								DFAgentDescription[] result = 
DFService.search(myAgent, template);

								merchantAgents = new AID[result.length];
								String merchantList = "";
								List<String> listStrings = new ArrayList<String>();

								merchantAgentsAcc = new AID[result.length];

								for (int i = 0; i < result.length; ++i) {
									merchantAgentsAcc[i] = 
result[i].getName();
								}

								for (int i = 0; i < merchantAgentsAcc.length; ++i) {

				listStrings.add(merchantAgentsAcc[i].getName().toString());
									if (i == 0) {
										merchantList = 
merchantAgentsAcc[i].getName().toString();
									} else {
										merchantList = merchantList + 
"@@" + merchantAgentsAcc[i].getName().toString();
									}

								}
								if (merchantList != null) {
									// The requested item is available for sale. 
Reply with the price

				reply.setPerformative(ACLMessage.PROPOSE);
									reply.setContent(merchantList);
								} else {
									// The requested book is NOT available for 
sale.
									// 
msg.setPerformative(ACLMessage.REFUSE);

				reply.setPerformative(ACLMessage.REFUSE);
									reply.setContent("not-available");
								}
								myAgent.send(reply);

							} catch (FIPAException fe) {
								fe.printStackTrace();
							}
						}
					} else {
						block();
					}
				}
			}
}
```

## Initiate customer order request:

```
@RequestMapping( method = RequestMethod.POST, params = "isrt")
	public String saveIntroducer(Model model, @ModelAttribute("newForm") CustOrderModel custOrderModel, BindingResult result , 
HttpServletRequest request) {

		HttpSession sess = request.getSession();
		Customerorder customerorder = new Customerorder();
		customerorder.setLatitude(custOrderModel.getLatitude());
		customerorder.setLongitude(custOrderModel.getLongitude());
```

```
                    customerorder.setOrdrProduct(custOrderModel.getOrderProdut());
                    customerorder.setQuantity(custOrderModel.getQuantity());
                    customerorder.setUserCredentialsId(custOrderModel.getUserCredentialsId());
                    Customer customer = custOrderDao.getCustomer(custOrderModel.getUserCredentialsId());

                            customerorder.setFirstName(customer.getFirstName());
                            customerorder.setLastName(customer.getLastName());
                            customerorder.setPhone(customer.getPhone());
                            customerorder.setEmail(customer.getEmail());


                    String errMsg = "";
                    String errMsgSts = "N";
                    Customerorder custOrder = new Customerorder();
                    custOrder= custOrderDao.insertRec(customerorder);
                    int resultOrder = custOrder.getId();

                    if (resultOrder >= 0){

                            model.addAttribute("ErrorMsgDisplay", "none");
                            model.addAttribute("orderId", resultOrder);

                model.addAttribute("successMsgDisplay", "Order request " + resultOrder + "successfully sent to the merchnat");
                model.addAttribute("successMsg", "Order request " + resultOrder + "successfully sent to the merchnat");

                    }
                    else{
                            model.addAttribute("ErrorMsgDisplay", "Error in request Order");

                model.addAttribute("successMsgDisplay", "none");

                    }

                    List<ProdCat> prodCat = ProdCatDao.getSelectCat();
                    model.addAttribute("prodCat", prodCat);
        model.addAttribute("newForm", new CustOrderModel());

                    return "od";

                              }
```

# Appendix 02:

## Genetic Algorithm process for selecting suitable merchants:

```
package lk.ac.mrt.msc.ai.masbe.Genetic;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
import static java.util.Objects.requireNonNull;
import java.util.function.Function;
import org.jenetics.BitGene;
import org.jenetics.engine.Codec;
import org.jenetics.engine.Engine;
import org.jenetics.engine.EvolutionResult;
import org.jenetics.engine.Problem;
import org.jenetics.engine.codecs;
import org.jenetics.util.ISeq;

/**
 * @author udayanga
 *
 */
public class MerchantSelectionModule implements Problem<ISeq<MerchantSelectionModule.Record>, BitGene, Double> {

	public static final class Record {
		final int id;
		final int category;
		final String name;
		final double distance;
		final double rating;
		final double latitude;
		final double longitude;
		final String discntStatus;

		public Record(final int id, final int category, final String name, final double distance, final double rating,
				final double latitude, final double longitude, final String discntStatus) {
			this.id = id;
			this.name = requireNonNull(name);
			this.category = category;
			this.distance = distance;
			this.rating = rating;
			this.latitude = latitude;
			this.longitude = longitude;
			this.discntStatus = discntStatus;
		}
	}

	public static final class ResultRecord {

		public String nameOfmerchant;
		public String merchantId;
		public String latitude;
		public String longitude;
		public String discntStatus;

		public ResultRecord(final String nameOfmerchant, final String merchantId, final String latitude,
				final String longitude, final String discntStatus) {

			this.nameOfmerchant = nameOfmerchant;
			this.merchantId = merchantId;
			this.latitude = latitude;
			this.longitude = longitude;
			this.discntStatus = discntStatus;
```

```
                }

        }

        private final ISeq<Record> _records;
//private final double _minMonthlyIncome;
//private final double _totalQaHours;
        private final double _totalMerchantsConsider;

        public MerchantSelectionModule(final ISeq<Record> records, final int noOfMerchantsConsider) {
                _records = requireNonNull(records);
                _totalMerchantsConsider = noOfMerchantsConsider;
        }

        public Function<ISeq<Record>, Double> fitness() {
                return records -> {

                        final double totalEvalFactor = records.stream().mapToDouble(r -> r.distance * (6 - r.rating)).sum();
                        final long totalCount = records.stream().mapToDouble(r -> r.id).count();
                        System.out.println("++fitness func");
                        return ((totalEvalFactor <= _totalMerchantsConsider * 3 * 3)) ? totalCount : 0.0; // +++ 3shd replace with

                                                                                                  // avg distance
                };
        }

        @Override
        public Codec<ISeq<Record>, BitGene> codec() {
                return codecs.ofSubSet(_records);
        }

        public ArrayList<ResultRecord> selectionProcess(ArrayList<Record> inputdata, int noOfMerchants) {
                ArrayList<ResultRecord> resultSet = new ArrayList<ResultRecord>();
                List<ResultRecord> resultCur = null;
                final int noOfMerchantsConsider = noOfMerchants;

                final MerchantSelectionModule merSelectionProcess = new MerchantSelectionModule(ISeq.of(inputdata),

                                        noOfMerchantsConsider);

                final Engine<BitGene, Double> engine = Engine.builder(merSelectionProcess).build();
                final ISeq<Record> result = merSelectionProcess.codec().decoder()
                                        .apply(engine.stream().limit(1000).collect(EvolutionResult.toBestGenotype()));

                final double totalEvalFactor = result.stream().mapToDouble(r -> r.distance * (6 - r.rating)).sum();

                final long curId = result.stream().mapToDouble(r -> r.id).count();
                System.out.println("*****curId:  " + curId);

                System.out.println("totalEvalFactor:  " + totalEvalFactor);

                System.out.println("Records:      " + result.map(r -> r.name).toString(", "));
                System.out.println("result.map(r -> r.id).toString() :" + result.map(r -> r.id).toString(", "));
                System.out.println("result.map(r -> r.distance).toString(): " + result.map(r -> r.distance).toString(", "));
                System.out.println("result.map(r -> r.rating).toString()) :" + result.map(r -> r.rating).toString(", "));
                // ResultRecord resultRecord = new ResultRecord(Integer.parseInt(result.map(r ->
                // r.id).toString()) , (result.map(r -> r.name).toString(", ")),
                // Double.parseDouble(result.map(r -> r.distance).toString()),
                // Double.parseDouble(result.map(r -> r.rating).toString()));
                ResultRecord resultRecord = new ResultRecord(result.map(r -> r.name).toString(","),
                                        result.map(r -> r.id).toString(","), result.map(r -> r.latitude).toString(","),
                                        result.map(r -> r.longitude).toString(","), result.map(r -> r.discntStatus).toString(","));
                resultSet.add(resultRecord);
                return resultSet;
        }

}
```

## Appendix 03:

```
package lk.ac.mrt.msc.ai.masbe.core.controller;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


import io.micrometer.core.annotation.Timed;
import lk.ac.mrt.msc.ai.masbe.core.model.Customerorder;
import lk.ac.mrt.msc.ai.masbe.core.service.OrderService;


import java.util.Collection;
import java.util.Optional;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


/**
 * @author udayanga
 *
 */
@RestController
@RequestMapping(value = "/custOrder")
@CrossOrigin(origins = "*")
```

```
@Timed
public class CustomerorderController {

        private static final Logger logger = LoggerFactory.getLogger(Customerorder.class);


        private final OrderService service;

  @Autowired
        public CustomerorderController(OrderService pservice) {
                this.service = pservice;
        }


        @GetMapping(value="/all")
         public ResponseEntity<Collection<Customerorder>> getAllProducts() {
          try {
            Optional<Collection<Customerorder>> customerorder = service.findAll();

            System.out.println("all fetched");

            int product_size = customerorder.get().size();
            if (product_size > 0) {
              logger.info("Number of products types found={}", product_size);
              customerorder.get().forEach((trxtype) -> {
                logger.info("Found:{}", customerorder);
              });
              return new ResponseEntity<>((Collection<Customerorder>) customerorder.get(), HttpStatus.OK);
            }
            else {
              logger.warn("No products types Found");
              return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }
          }
          catch (Exception e) {
            logger.error("{}", e);
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
          }
          finally {
          }
         }

        @GetMapping(value = "/{id}")
         public ResponseEntity<Customerorder> getProductById(@PathVariable int id) {
          try {
            Optional<Customerorder> customerorder = service.findById(id);
            if (customerorder.isPresent()) {
              logger.info("order type with id={} found:{}", id, customerorder.get());
              return new ResponseEntity<>(customerorder.get(), HttpStatus.OK);
            }
            else {
              logger.warn("Transaction type with id={} not found", id);
              return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }
          }
          catch (NumberFormatException nfe) {
            logger.warn("{}", nfe);
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
          }
          catch (Exception e) {
            logger.error("{}", e);
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
          }
          finally {
          }
         }
```

```
@PostMapping

ResponseEntity<?> addOrder(@RequestBody Customerorder newOrder) {

  try {

   // Optional<Customerorder> customerorder = service.findById(newOrder.getId());

              System.out.println(" ORDER ID : "+ newOrder.getId());

    if (newOrder.getId()== 0) {

      logger.info("new OrdeR  Saving to database.", newOrder.getId(), newOrder);

      service.addOrder(newOrder);

      return new ResponseEntity<>(HttpStatus.CREATED);

    }

    else {

      logger.warn(" New Order not saving to database.", newOrder.getId());

      return new ResponseEntity<>(HttpStatus.CONFLICT);

    }

  }

  catch (Exception e) {

    logger.error("{}", e);

    return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);

  }

  finally {

  }

}


@PutMapping(value = "/{id}")

public ResponseEntity<Customerorder> updateOrder(@PathVariable(required = true, value = "id") int id, @RequestBody
Customerorder custOrder) {

  try {

    if (service.exists(id)) {

      logger.info("customer order update with id={} found. Updating database the database with {}.", custOrder.getId(), custOrder);

      custOrder.setId(id);

      service.updateOrder(custOrder);

      return new ResponseEntity<>(HttpStatus.OK);

    }
  else {
    logger.warn("customer order with id={} not found. Not updating database.", custOrder.getId());
    return new ResponseEntity<>(HttpStatus.NOT_FOUND);
  }
}
catch (Exception e) {
  logger.error("{}", e);
  return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
}
finally {
}
```

## Appendix 04:

Customer Order Table:

```
CREATE TABLE `customerorder` (
`id` INT(11) NOT NULL AUTO_INCREMENT,
`first_name` VARCHAR(45) NULL DEFAULT NULL,
`last_name` VARCHAR(45) NULL DEFAULT NULL,
`email` VARCHAR(45) NULL DEFAULT NULL,
`phone` VARCHAR(15) NULL DEFAULT NULL,
`profile_pic` LONGBLOB NULL,
`user_credentials_id` INT(11) NOT NULL,
`notify_via_email` VARCHAR(1) NOT NULL DEFAULT 'Y',
`notify_via_sms` VARCHAR(1) NOT NULL DEFAULT 'Y',
`streaming_payment_allowed` VARCHAR(1) NULL DEFAULT 'Y',
`status` VARCHAR(1) NOT NULL DEFAULT 'I',
`nic` VARCHAR(45) NULL DEFAULT NULL,
`passport` VARCHAR(45) NULL DEFAULT NULL,
`created_date` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
`last_modified_date` DATETIME NULL DEFAULT NULL,
`ordrStatus` VARCHAR(1) NULL DEFAULT 'A',
`latitude` FLOAT(10,6) NULL DEFAULT NULL,
`longitude` FLOAT(10,6) NULL DEFAULT NULL,
`ordrProduct` VARCHAR(30) NULL DEFAULT NULL,
`ordrProductType` INT(11) NULL DEFAULT NULL,
`quantity` INT(11) NULL DEFAULT '1',
`buyMerchantId` INT(11) NULL DEFAULT NULL,
`merchant_offerprice` FLOAT(10,6) NULL DEFAULT NULL,
`order_accept_status` VARCHAR(1) NULL DEFAULT 'I',
`customer_offerprice` FLOAT(10,6) NULL DEFAULT NULL,
`customer_rejMerchant` INT(11) NULL DEFAULT NULL,
PRIMARY KEY (`id`)
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
AUTO_INCREMENT=33
;
```

Merchant Inventory Table:

```
CREATE TABLE `merchantinventory` (
`id` INT(11) NOT NULL AUTO_INCREMENT,
`prod_cat` INT(11) NULL DEFAULT NULL,
`prod_code` VARCHAR(45) NOT NULL,
`merchant_id` INT(11) NOT NULL,
`prod_name` VARCHAR(100) NULL DEFAULT NULL,
`price` DECIMAL(13,2) NOT NULL,
`quantity` INT(11) NOT NULL,
`status` VARCHAR(1) NOT NULL DEFAULT 'A',
`DiscntRate` FLOAT(10,6) NULL DEFAULT NULL,
PRIMARY KEY (`id`),
INDEX `fk_merchant_order_merchant1_idx` (`merchant_id`),
INDEX `fk_merchant_prod_cat` (`prod_cat`),
CONSTRAINT `fk_merchant_order_merchant1` FOREIGN KEY (`merchant_id`)
REFERENCES `merchant` (`id`) ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT `fk_merchant_prod_cat` FOREIGN KEY (`prod_cat`) REFERENCES
`prodcat` (`id`) ON UPDATE CASCADE ON DELETE CASCADE
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
AUTO_INCREMENT=13
```

Merchant Loyalty:

```
CREATE TABLE `merchantcustloyality` (
`id` INT(11) NOT NULL AUTO_INCREMENT,
`merchant_id` INT(11) NOT NULL,
`cust_id` INT(11) NOT NULL,
`points` INT(11) NOT NULL,
`status` VARCHAR(1) NOT NULL DEFAULT 'A',
PRIMARY KEY (`id`),
INDEX `fk_merchantCustLoyality` (`merchant_id`),
INDEX `fk_customer_user_loyality_idx` (`cust_id`),
CONSTRAINT `fk_customer_user_loyality` FOREIGN KEY (`cust_id`) REFERENCES
`user_credentials` (`id`) ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT `fk_merchantCustLoyality` FOREIGN KEY (`merchant_id`) REFERENCES
`merchant` (`id`) ON UPDATE NO ACTION ON DELETE NO ACTION
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
AUTO_INCREMENT=3
;
```

Merchant:

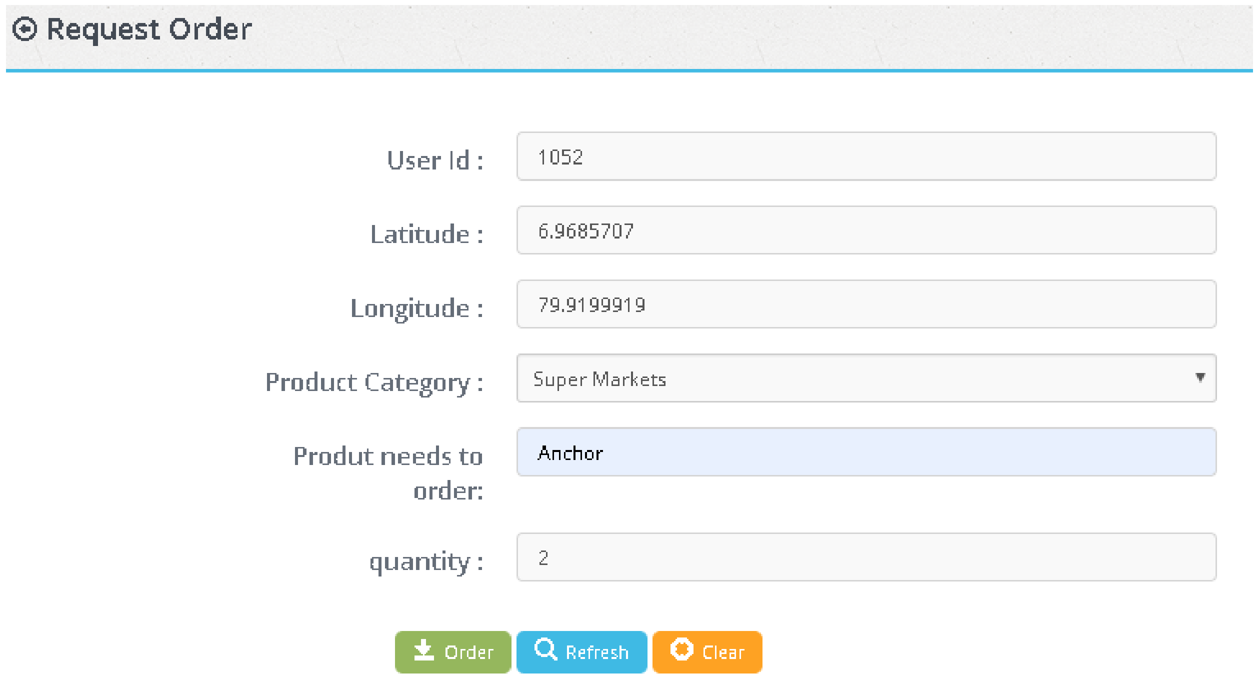
```
CREATE TABLE `merchant` (
`id` INT(11) NOT NULL AUTO_INCREMENT,
`name` VARCHAR(100) NULL DEFAULT NULL,
`code` VARCHAR(45) NULL DEFAULT NULL,
`description` VARCHAR(250) NULL DEFAULT NULL,
`ipay_account_id` INT(11) NOT NULL,
`logo` LONGBLOB NULL,
`merchant_category_id` INT(11) NOT NULL,
`business_reg_no` VARCHAR(45) NULL DEFAULT NULL,
`business_email` VARCHAR(45) NULL DEFAULT NULL,
`business_mobile` VARCHAR(45) NULL DEFAULT NULL,
`status` VARCHAR(1) NULL DEFAULT 'A',
`streaming_payment_allowed` VARCHAR(1) NULL DEFAULT 'N',
`merchant_type_id` INT(11) NOT NULL,
`br_certificate` LONGBLOB NULL,
`refund_policy` VARCHAR(1000) NULL DEFAULT NULL,
`address_line1` VARCHAR(100) NULL DEFAULT NULL,
`address_line2` VARCHAR(100) NULL DEFAULT NULL,
`city` VARCHAR(45) NULL DEFAULT NULL,
`nic` VARCHAR(20) NULL DEFAULT NULL,
`nic_image` LONGBLOB NULL,
`telephone_no` VARCHAR(45) NULL DEFAULT NULL,
`api_token` VARCHAR(200) NULL DEFAULT NULL,
`developer_enabled` VARCHAR(1) NULL DEFAULT 'N',
`subscriber_user` VARCHAR(45) NULL DEFAULT NULL,
`reference_code` VARCHAR(45) NULL DEFAULT NULL,
`created_date` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
`created_by` VARCHAR(45) NULL DEFAULT NULL,
`last_modified_date` DATETIME NULL DEFAULT NULL,
`last_modified_by` VARCHAR(45) NULL DEFAULT NULL,
`enrollment_type` VARCHAR(45) NULL DEFAULT NULL,
`is_virtual_merchant` VARCHAR(1) NULL DEFAULT 'N',
`is_search_and_pay_enabled` VARCHAR(1) NULL DEFAULT 'Y',
`acquired_bank_id` INT(11) NULL DEFAULT '0',
`emv_merchant_code` VARCHAR(45) NULL DEFAULT NULL,
`callback_url` VARCHAR(1000) NULL DEFAULT NULL,
`introducer_id` INT(11) NOT NULL DEFAULT '0',
`ordrStatus` VARCHAR(1) NULL DEFAULT 'I',
`latitude` FLOAT(10,4) NULL DEFAULT NULL,
`longitude` FLOAT(10,4) NULL DEFAULT NULL,
`DiscntStatus` VARCHAR(1) NULL DEFAULT 'I',
PRIMARY KEY (`id`),
UNIQUE INDEX `reference_code_UNIQUE` (`reference_code`),
UNIQUE INDEX `emv_merchant_code_UNIQUE` (`emv_merchant_code`),
      INDEX `fk_merchant_ipay_account1_idx` (`ipay_account_id`)
);
```

Customer:

```
CREATE TABLE `customer` (
`id` INT(11) NOT NULL AUTO_INCREMENT,
`first_name` VARCHAR(45) NULL DEFAULT NULL,
`last_name` VARCHAR(45) NULL DEFAULT NULL,
`email` VARCHAR(45) NULL DEFAULT NULL,
`phone` VARCHAR(15) NULL DEFAULT NULL,
`profile_pic` LONGBLOB NULL,
`user_credentials_id` INT(11) NOT NULL,
`notify_via_email` VARCHAR(1) NOT NULL DEFAULT 'Y',
`notify_via_sms` VARCHAR(1) NOT NULL DEFAULT 'Y',
`streaming_payment_allowed` VARCHAR(1) NULL DEFAULT 'Y',
`status` VARCHAR(1) NOT NULL DEFAULT 'I',
`nic` VARCHAR(45) NULL DEFAULT NULL,
`passport` VARCHAR(45) NULL DEFAULT NULL,
`created_date` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
`last_modified_date` DATETIME NULL DEFAULT NULL,
PRIMARY KEY (`id`),
INDEX `fk_customer_user_credentials1_idx` (`user_credentials_id`),
CONSTRAINT `fk_customer_user_credentials1` FOREIGN KEY
(`user_credentials_id`) REFERENCES `user_credentials` (`id`) ON UPDATE NO
ACTION ON DELETE NO ACTION
)
```

## Appendix 05:

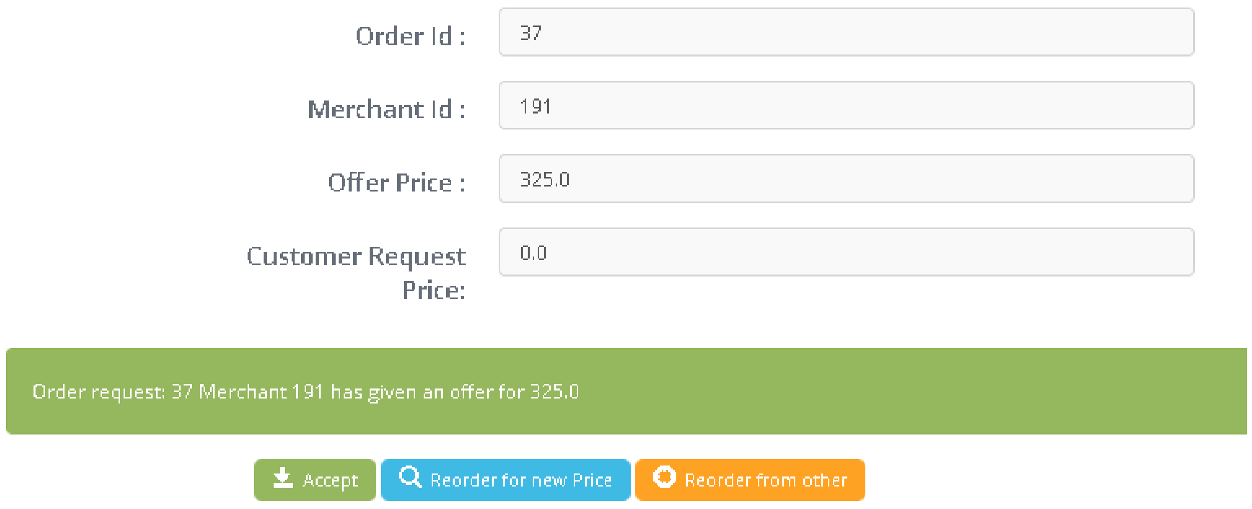
### Request Order UI



Request Order

| | |
|---|---|
| User Id : | 1052 |
| Latitude : | 6.9685707 |
| Longitude : | 79.9199919 |
| Product Category : | Super Markets |
| Produt needs to order: | Anchor |
| quantity : | 2 |

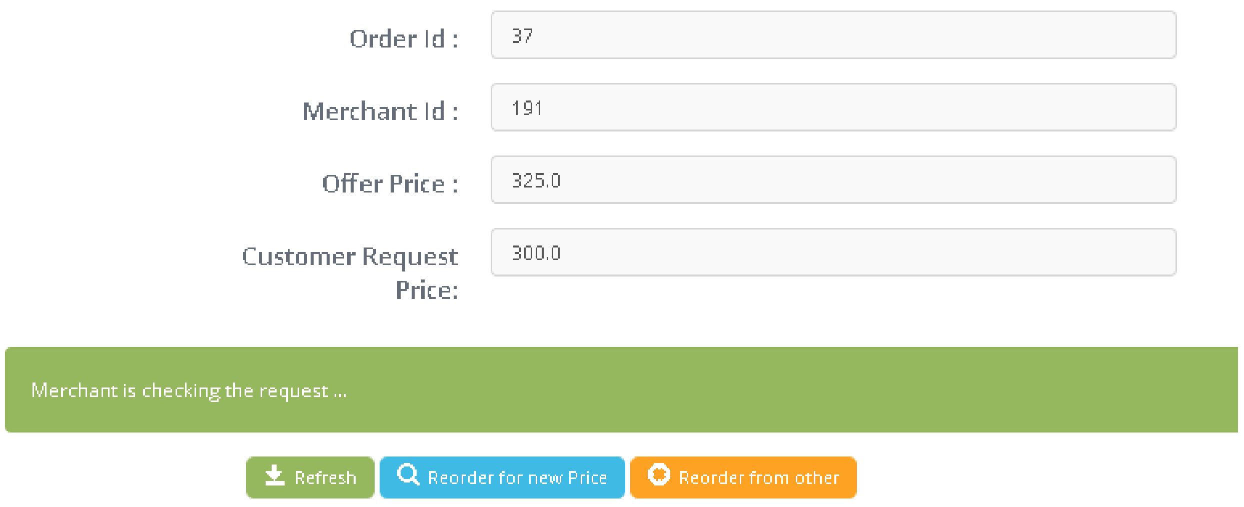
### Merchant offer UI



| | |
|---|---|
| Order Id : | 37 |
| Merchant Id : | 191 |
| Offer Price : | 325.0 |
| Customer Request Price: | 0.0 |

Order request: 37 Merchant 191 has given an offer for 325.0

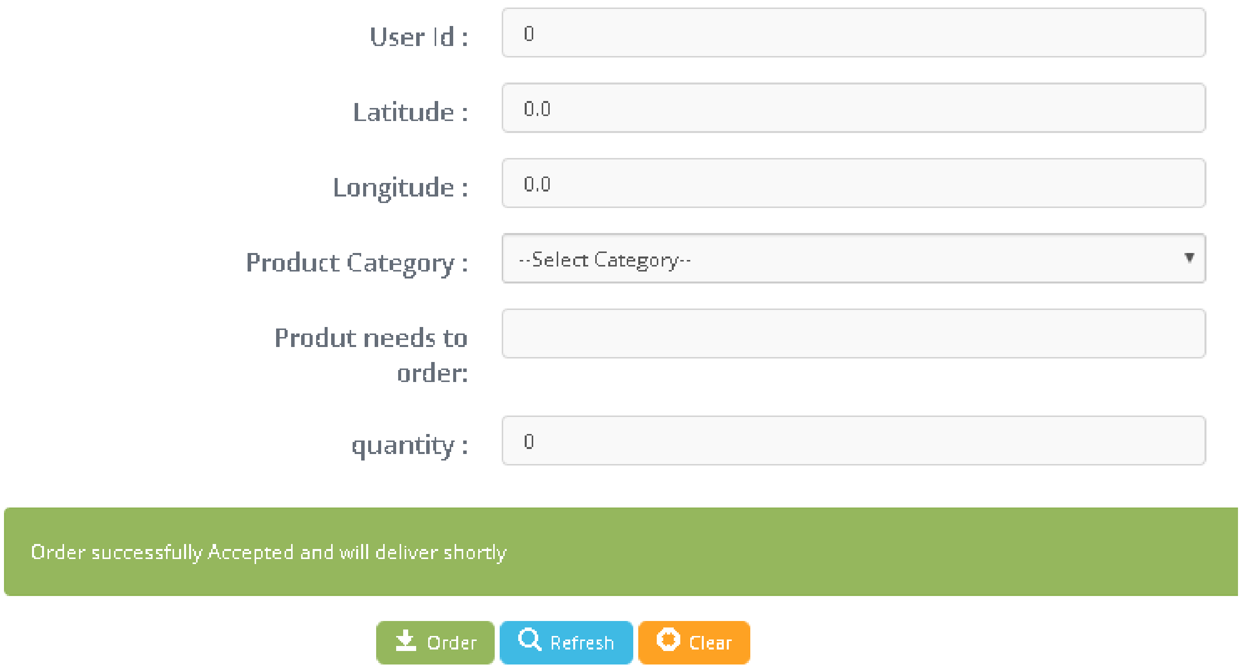
Customer offer UI



| | |
|---|---|
| Order Id : | 37 |
| Merchant Id : | 191 |
| Offer Price : | 325.0 |
| Customer Request Price: | 300.0 |

Merchant is checking the request ...

Order Acceptance UI



| | |
|---|---|
| User Id : | 0 |
| Latitude : | 0.0 |
| Longitude : | 0.0 |
| Product Category : | --Select Category-- |
| Produt needs to order: | |
| quantity : | 0 |

Order successfully Accepted and will deliver shortly