AGENT BASED METERING SYSTEM FOR ENERGY NETWORKS

Dulan Maheeka Diggaha Ranawaka

109247J

Dissertation Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Science in Electrical Engineering

Department of Electrical Engineering

University of Moratuwa

Sri Lanka

June 2015

DECLARATION

"I declare that this is my own work and this dissertation 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 dissertation, 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)".

…… ……

Signature of the candidate Date

(D.M.D. Ranawaka)

The above candidate has carried out research for the Masters Dissertation under my supervision.

……..... ……..

Signature of the supervisor Date

(Dr. K.T.M.U. Hemapala)

ABSTRACT

Centralized automated single server based energy meter reading systems required high-end resources, at the same time it has low efficiency benefits, low reliability and real time meter data monitoring get more complex when the connected meter base is large.

Therefore objective of the research is to provide a methodological approach for energy meter reading systems to apply for the Sri Lankan context using de-centralized technique. Agent based solution was implemented with Multi Agent System (MAS) to address above issues and the system is characterized de-centralized nature and self configurable nature.

Low cost, high reliable, high efficient meter reading system is provided by MAS based decentralized system

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude to my supervisor Dr. K.T.M.U Hemapala and Dr. P.S.N De Silva, for their continuous guidance, constructive feedback and support extended throughout this research. Despite their busy schedule, they always remained accessible and their guidance and advice, expertise and insights were by all means truly invaluable.

I am grateful to my parents for all the sacrifices they have made along the way and for all the support given to me during various endeavours in my life. Finally, I would like to sincerely thank my wife Soniya for being supportive and understanding, and for all her love and encouragement throughout this research.

D.M.D Ranawaka

June 2015

CONTENT

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Description**
AMS	Agent Monitoring System
API	Application Program Interface
ASM	Anchor Security Manager
AJNDI	Anchor Java Naming, Directory Interface
BSD	Berkeley Source Distribution
COSEM	Companion Specification For Energy Metering
CT	Current Transformer
DB	Database
DLMS	Device Language Message Specification
DF	Directory Facilitator
FIPA	Foundation For Intelligent Physical Agents
GUI	Graphical User Interface
IEC	International Electro-Technical Commission
IP	Internet Protocol
JADE	Java Agent Development Framework
KQM	Knowledge Query Manipulation Language
MAS	Multi Agent Systems
OBIS	Object Identification System
PC	Personal Commuter
PDA	Personal Digital Assistant
RMI	Remote Monitoring Interface
SATP	Secure Agent Transfer Protocol
TCP	Transmission Control Protocol
VCC	Virtual Code Compiler

LIST OF APPENDIX

1 INTRODUCTION

1.1 Multi Agent Systems

Multi Agent Systems (MAS) become more and more powerful for real world applications which must meet the demanding need of higher and increasing complexity as well as retaining the integrity of the system. To achieve such requirements, it becomes almost impossible for traditional software design techniques to maintain the integrity of the system as well as keeping the functionality of the system as needed. Multi Agent Systems is a developing field of software where it becomes possible to achieve for such difficult tasks. MAS allow us to do tasks which were previously impossible or impracticable with well known software designs.[1]

As a software application more complex, it becomes more and more vulnerable to failure as a result of component breakdowns. Thus it requires extreme amounts of attention to control the growth of the application in order to maintain the integrity of the system. However, in real world application, the situation differs widely as it becomes ever more difficult to predict the component which may fail in advance. It becomes highly productive to reduce the amount of work done by a single component of a system. If explained further, the amount of task specialization should be at a minimum, and the logic of a certain task should be distributed throughout the system.

So, one would question the purpose of distributing the application logic and not dedicating a separate component to do a specific task. Reasons are as follows,[1][2]

i) As mentioned, when a system grows, it becomes highly subjected to errors and the frequency of error encounter increases drastically. But such large systems have a high degree of importance to the people. Therefore, a failure of a critical resource of the system may lead a very high number of people without the availability of the application.

ii) When a component is dedicated a special task, all actions which require the service of that dedicated component must query that server in order to accomplish the task. If that dedicated service cannot be catered the input at the desired rate, this dedicated service becomes an application bottleneck. Thus reducing the total application's productivity.

iii) If the scenario is further analyzed, it would be found that such a service cannot be constantly increased without the upgrading of critical service components. Eg: Suppose a Database (DB) server. If the number of request to the DB server is increased by a huge value, it needed to distribute the load amongst another DB server to cater the application demand. Otherwise, it would not be able to get the required performance. Therefore, the system cannot be expanded without caring about the critical services improvement.

1.2 Multi Agent Systems for Meter Reading

When meter reading and analysing systems considered, it required highly complex systems which require to real time operation. If the practicability of using MAS for such complex meter reading systems considered, it becomes clear that the uses of MAS functionality gives greater flexibility to handle such difficult task.

1.3 Overall Objective

Overall objective is to build a Multi Agent System which would enable us to have some collaboration with energy meters, as a whole, to read all meters real time with more reliable and more efficient manner. In other words, distributing the meter reading task in to multi-distributed systems in dynamic manner and getting the result simultaneously is aimed.

1.4 Can Theoretically Reliable System be Achieved with MAS?

According to the features of MAS, even though single component is failed, the whole system is get affected, because the duties of failed component is taken over

by another similar working component in run time. In other words reliability of the whole system get improved.

1.5 Can Theoretically Efficient System be Achieved with MAS?

According to the features of MAS, the whole task is distributed among several smaller components multi-located in network. As each component has a limited task and they are operated simultaneously, it reduces the time to carry out whole task. In other words efficiency of the whole system get improved.

1.6 Would Associated Cost Become an Issue?

When a power distribution utility is concerned, there are many offices located everywhere in its operating region. The several small agent components can be deployed into the PCs which are already been used without introducing any additional cost to the system In other words any dedicated high-end resources are not required to run the system.

2 LITERATURE REVIEW

2.1 Principal Approaches of Meter Reading System

There are several approaches to develop real time meter reading systems. This chapter reviews the different technological approach and their importance and the performance.

2.1.1. Centralized approach

Basically the centralized system works as a simple client/server mechanism. The Client/server computing systems are comprised of two logical parts. A server that provides services and a client that requests services from the server. Together, the two form a complete computing system with a distinct division of responsibility. More technically, client-server computing relates two or more threads of execution using a consumer producer relationship. Clients serve as the consumers in a client-server system. That is, they make requests to servers for services or information and then use the response to carry out their own purpose. The server plays the role of the producer, filling data or service requests made by clients [3],[4]. In a centralized structure one or five servers running central location and the meter data is fed to the servers for processing. The centralized system architecture is illustrated in figure 2.1. This type of system is well developed system for small scale power system. When the distribution system becomes large, following drawbacks to be faced.[5]

- The main sever system is expensive since high processing power with high-end resources are required.
- Software which has high sophisticated optimization algorithm is required as system optimization software.
- Expensive fast communication systems need be used.
- Additional cost on good redundancy system.
- The entire system can be failed as a result of single point break down

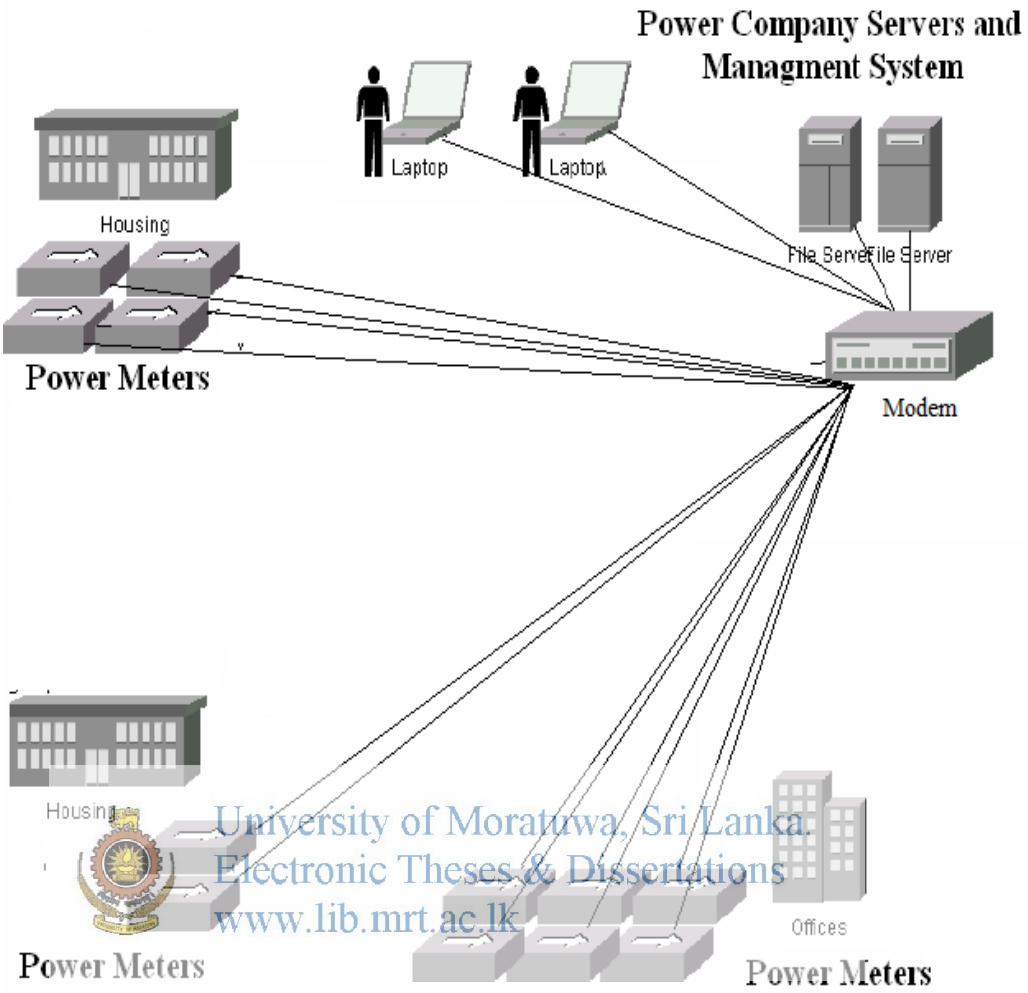


Figure 2.1: Centralized Meter Reading System Approach

2.1.2. Decentralized approach

Basically it is more similar to centralized approach. For a large scale meter base, meters can be clustered into several groups and download meter readings separately and after processing, processed data will be sent to the central monitoring system. In Figure 2.2 it illustrates the decentralized system architecture. This type of system is well developed system for medium scale power system. When the distribution system becomes large, following drawbacks to be faced.

- The main sever system is expensive since high processing power with high-end resources are required.

- Software which has high sophisticated optimization algorithm is required as system optimization software
- Expensive fast communication systems needed be used between main monitoring system and sub systems.
- Additional cost on good redundancy system.
- Single failure in a system lead to total or partial failure.

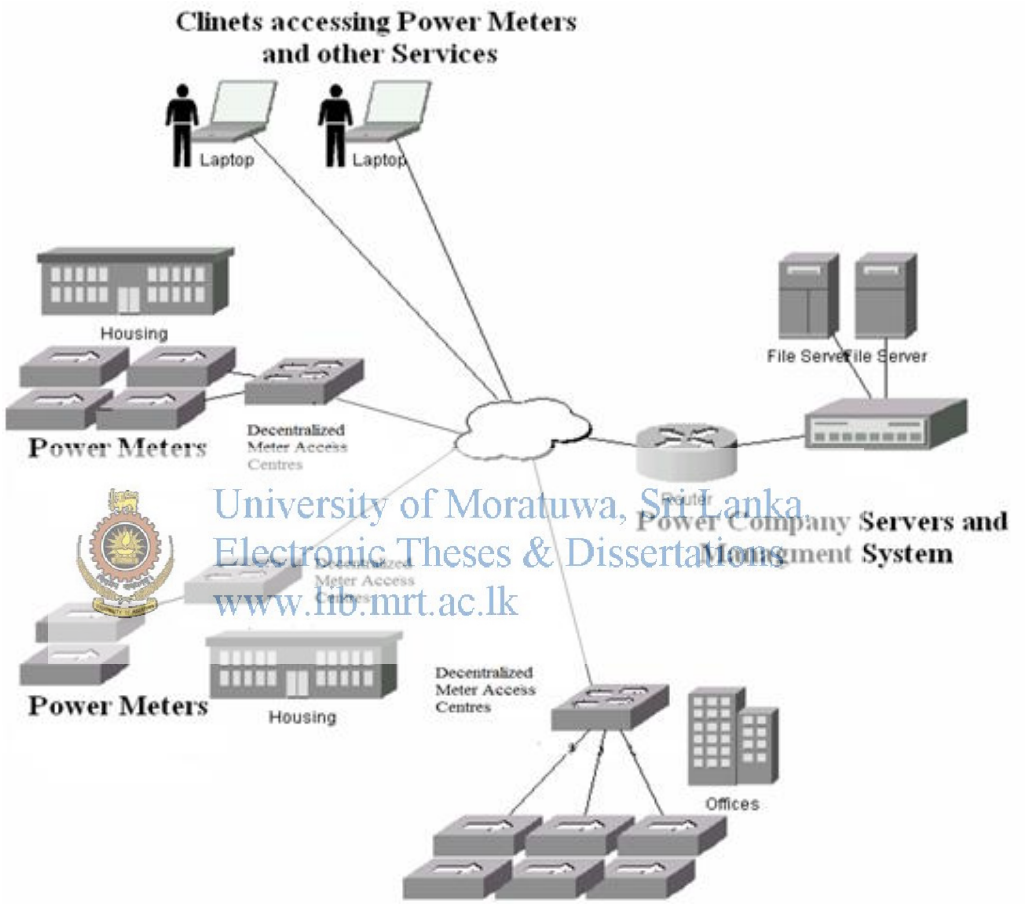


Figure 2.2: Decentralized Meter Reading System Approach

2.1.3. Mobile agent approach

A mobile agent is an executing program that can migrate during execution from machine to machine in a heterogeneous network. On each machine, the agent interacts with stationary service agents and other resources to accomplish its task. Mobile agents are particularly attractive in distributed information retrieval

applications.[6] By moving to the location of an information resource, the agent can search the resource locally, eliminating the transfer of intermediate results across the network and reducing end-to-end latency. Mobile agents as described here visiting each meter and executed on the embedded web server checking the status and reading the power consumption value, encapsulating this data and move to visit another power meter. In Figure 2.3 the mobile agent system architecture is illustrated. This type of system is well developed system for large scale power system. But when considering such system that the each meter should have embedded micro web-server and it should capable of hosting this mobile agent and has the required and enough functionality of handling the mobile agent's issues including security, fault tolerance and not cost effective to have such high-end meter [7][8].

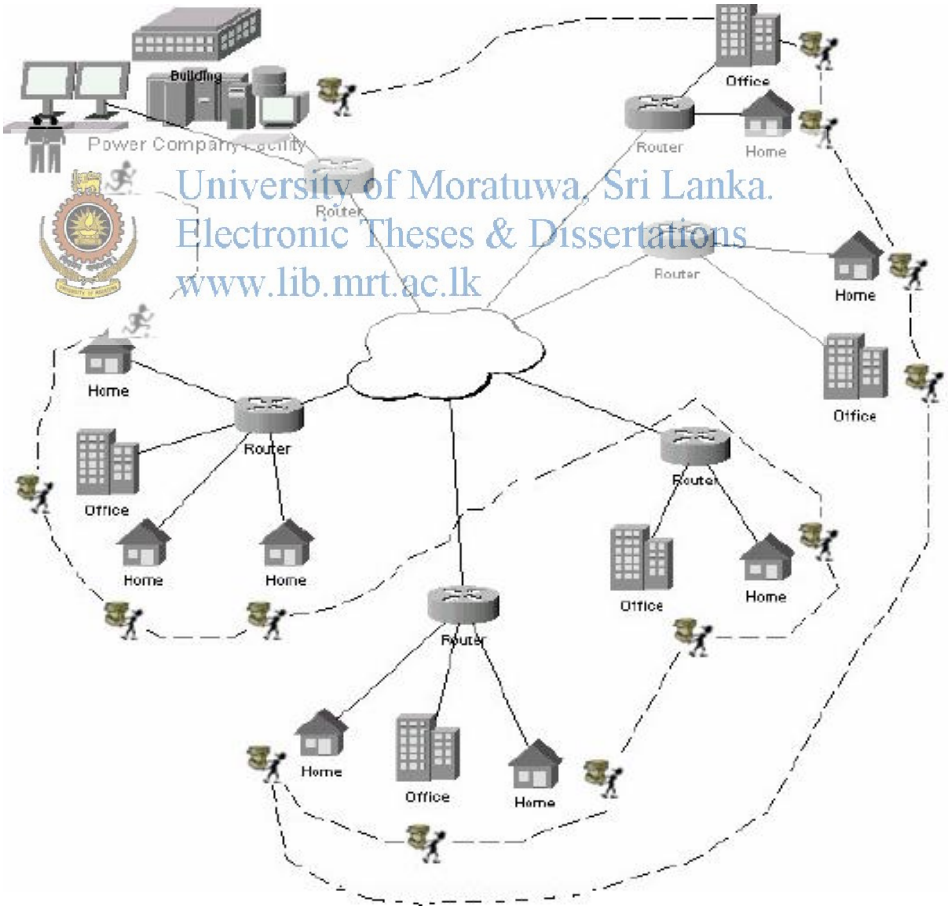


Figure 2.3 Mobile Agent Meter Reading System Approach

2.1.4. Available multi agent platforms and theory

Large scale implementation of agent systems requires frameworks, methodologies and tools that support effective development of agent systems. Historically the main obstacle in agent and multi agent system development used to be the infrastructure which refers to supporting environment where agents can communicate and achieve their desired goals.

Nowadays many agent development tools and platforms of different quality and maturity are available and have been employed for different applications. There is so far no consensus about which tool is best for agent development.

Various agent toolkits are analyzed in depth including their strengths and weaknesses.

2.1.4.1. IBM-Aglet

IBM-Aglet is an environment for developing mobile agents based application in JAVA. It is an open source freely available toolkit. It provides good graphical user interface for agent development. It mainly comprised of two packages The Aglet Building Environment and the Aglet Workbench.[9] Aglet workbench aims at developing stand alone mobile agents.

Aglets are basically java objects comprising of two major components namely Aglet Core and Aglet Proxy. Core is holder of all the internal variables and methods of an agent. Proxy acts as an interface to the core.[9]

Aglet server is an application program that works as agent server for aglets. It provides users with a good GUI and allows users to create an agent, monitor it and dispose it off when required. It gives user the ability to set agent's access privileges on the server. For an aglet to move to a remote host, it must have server installed on it, and it solves some of the security problems. The Agent migration is implemented using socket mechanism. Communication among agents is achieved using synchronous and asynchronous message passing.[10]

Although Aglet platform has wide user acceptance but it doesn't provide much security. Its security is knitted in the concept of restricting transfer of aglets only to

its own servers. Due to lack of security, state of aglets can't be stored on any other host.[11] No such method is provided by this tool. Scalability is another problem, since aglets are not interoperable with other platforms or their agents, due to their restriction of working with their own server.

2.1.4.2. Voyager

Voyager is a simple powerful technology for creating mobile agents in Java. It was an improvement over already existing platforms like Aglets, Odyssey, Concordia etc. which only allowed developers to create agents and launch them into a network to fulfil its mission. But none allowed sending messages to a moving agent, which made it difficult to communicate with an agent once it has been launched and also for agents to communicate with other agents.[12]

It treats an agent like a special kind of object which can move independently and can continue its execution while moving around. The agents can move autonomously in the network. It allows an agent to send and receive Java messages to and from other agent, even while traversing the network, irrespective of its position in the network. Whenever an agent moves, it leaves behind a forwarder object which forwards the message to its new location.[11]

Voyager provides flexible life spans for agents, the agent's life span can be changed flexibly as required.

- An agent can live until it has none local or remote references.
- Agent can live for a certain amount of time.
- It can live for a particular point in time.
- It can live until it remains inactive for a specified time.
- An agent can live forever.

Another attractive feature of this tool is its support for directory service, which is particularly important in launching a mobile agent from one application to another and for locating an agent after it moves to some other location in the network. Its directory structure allows creating and connecting network directories together to generate a large interlinked directory structure.

Voyager supports weak mobility of agents using Remote Monitoring Interface (RMI) technique. It allows objects to be mobile using Virtual Code Compiler (VCC). VCC utility accepts any .class or .java file and produces a new remote enabled virtual class. This virtual class is used in further communications with that agent. Agents use "moveTo()" function and a callback function for migrating to a remote host. On reaching new host, the agent retrieves the callback function that it sent and resumes its execution.[13]

Voyager has an associated server called 'voyager' but it's not necessary to have such server installed on all nodes in the network. Due to this reason agents created using voyager are provided restricted access on the host servers. Thus provision of agent and host security is weak in this tool.

2.1.4.3. Anchor

Anchor facilitates the transmission and secure management of mobile agents in heterogeneous distributed environments. This toolkit is having (Berkeley Source Distribution) BSD style license. Its architecture comprised of an agent viewer graphical user interface, Agent API, Anchor server, Anchor security manager (ASM) , Anchor class loader (ACL), secure agent transfer protocol (SATP) handler, Anchor Java Naming Directory Interface (AJNDI) and Anchor Java Native Interface (AJNI) components. In Figure 2.4 shows the architecture of the Anchor framework.[14]

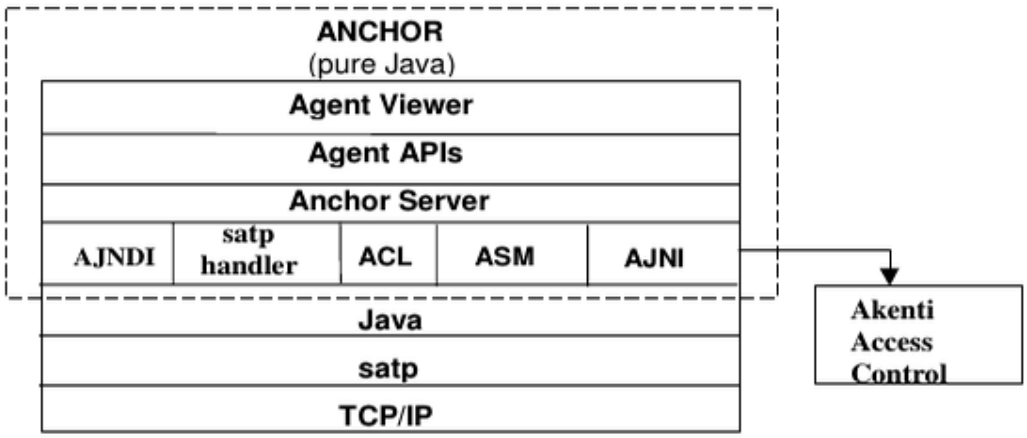


Figure 2.4 architecture of the Anchor framework

Agent model in Anchor is based on that of Aglets. Agents are java objects capable of migrating in the network. Agents are created in contexts, where context is a namespace under which agents are grouped together. Agents are accessed through their proxies, which protect the agent from any attempt of direct access to its code and methods. Also it provides location transparency to agent which means an agent is represented in a machine even if it has migrated to some other machine in the network. All messages to that agent are forwarded to its location through its proxy.

Agent server in Anchor is a run time environment which acts as backbone of this toolkit. It runs on a host and works on a specific port. It performs all system related functions. Anchor server supports the agent migration through SATP. Security is a major concern in this toolkit. Mutual authentication between agent systems is established through secure socket layer (SSL). Agents are authenticated by signing their byte codes with their private keys.

Akenti is an access control system designed to ensure controlled access of distributed resources. This component uses Public key infrastructure. Access control decisions are made using digitally signed certificated based on X.509 standard.

Integration of Akenti component in Anchor provides it strong security. Anchor provides a naming service through which every agent can register and publish its current information. It also provides a directory service to enable effective searching of agents. Agent viewer component supports features like creating an agent, its dispatch, retraction, disposal, activation and deactivation and also cloning of an agent.

2.1.4.4. Zeus

Zeus is an integrated environment for the rapid development of collaborative agent applications. It is open source freely available toolkit. It is purely implemented in Java which makes it compatible with most hardware platforms. It also complies with FIPA standards. In Figure 2.5 shows the architecture of the Zeus framework.

Zeus provides support for generic agent functionality and has sophisticated support for the planning and scheduling of an agent's actions. It provides a set of software

components and tools used to design, develop and organize agent systems. It has good graphical user interface and embedded components like report generation tool, statistical tool, agent and society viewer tool etc. which help in observation of application under development. Also it allows the designers to use different negotiation techniques for testing implemented agents.

Communication among agents is performed using Agent Communication Language (ACL) or Knowledge Query Manipulation language (KQML). Communication security is provided using public key, private key cryptography and digital signature technologies Major drawbacks of this toolkit include lack of support for agent mobility and its weak documentation which leads to difficulties in creation of new applications.[15]

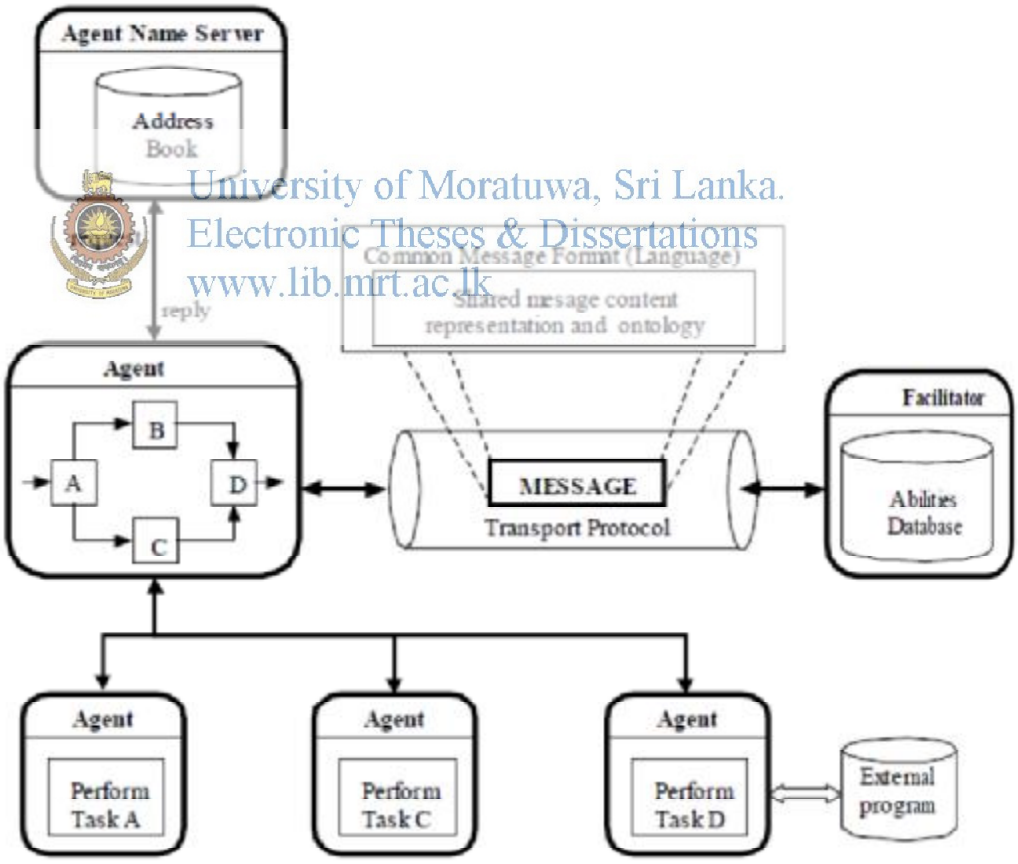


Figure 2.5: Architecture of the Zeus Framework.

2.1.4.1. JADE

JADE (Java Agent Development Framework) is a software Framework fully implemented in Java language. It is development of multi-agent applications based on peer-to-peer communication architecture. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the latest Foundation for intelligent physical agents (FIPA) specifications. It provides a set of graphical tools that supports the debugging and deployment phases of agent development. Jade permits the intelligence, information & resources to be distributed over the network in the form of java compatible mobile devices like PDA, pagers, cell phones, smart phones, laptops or fixed desktops etc. [1]

In JADE an instance of run-time environment is called a container, as it holds all the agents created in it. Collection of such containers is called a platform and it provides a homogenous layer which hides the complexity of underlying hardware & software from agents and their developers. It is compatible with J2ME, J2EE & CLDC. Its low memory requirements make it suitable for mobile devices. In Figure 2.6 illustrates the architecture of JADE framework.

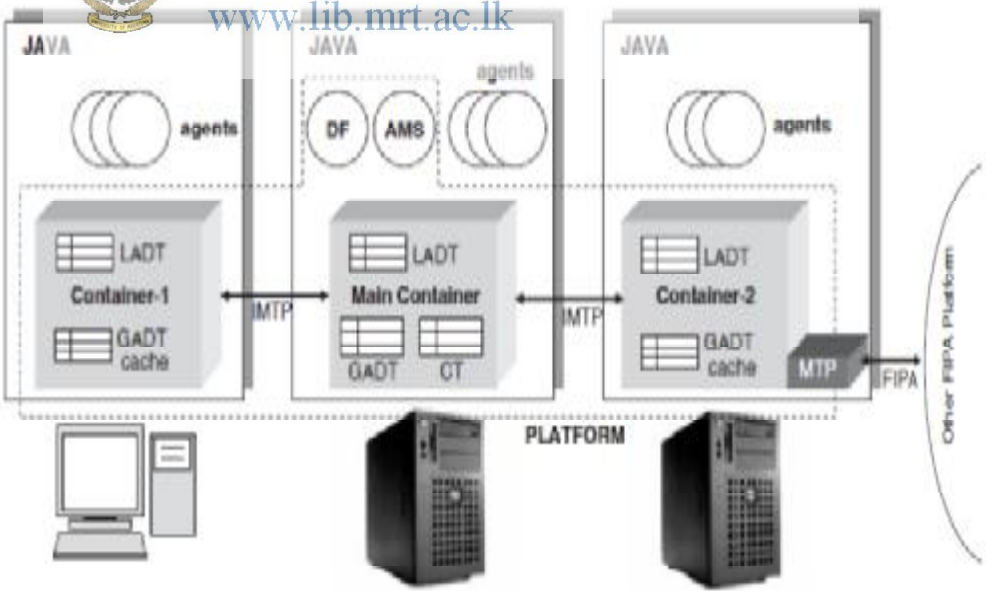


Figure 2.6: Architecture of the JADE Framework

Every container in JADE comprises of Directory Facilitator (DF) agent, Remote Monitoring Interface (RMI) agent & Agent Monitoring System (AMS) agent. Here the agents can discover other agents dynamically using DF agent & can communicate with each other using peer-to-peer paradigm.[16]

In JADE agents communicate using asynchronous message passing technique which is most widely accepted model for distributed and loosely coupled communications. JADE preserves the security of agents by providing strong authentication mechanisms to verify rights of any agent. Messages exchanged among agents comply with Agent Communication Language (ACL) defined by FIPA. JADE supports execution of multiple parallel tasks within the same java thread. This feature ensures scalability as well as meets resource constraints of environment.

It supports agent mobility by allowing an agent to transfer its code as well as its state to remote hosts. JADE Object Manager provides connection authentication, user validation and RPC message encryption. The Jade socket proxy agent acts as bidirectional gateway between a JADE platform and an ordinary TCP/IP connection. Interoperability is an attractive feature of JADE, since it is FIPA compliant. Thus agents created in JADE can interoperate with other agents, following the same standards. Also it supports complex interaction protocols like contract net protocol for facilitating complex multi-agent applications.[1]

3 SYSTEM DEVELOPMENT

3.1 Conceptual Design

The main components of the multi agent based meter reading system (MRS) are given in the Figure 3.1. As illustrated in the figure, the MAS consist of three types of agents namely Database Management Agents, Meter Access Agents and Meter Monitor Agents. The combination of those three types of agents creates the multi agent system application and it communicates with the physical meters via wireless GPRS connectivity.

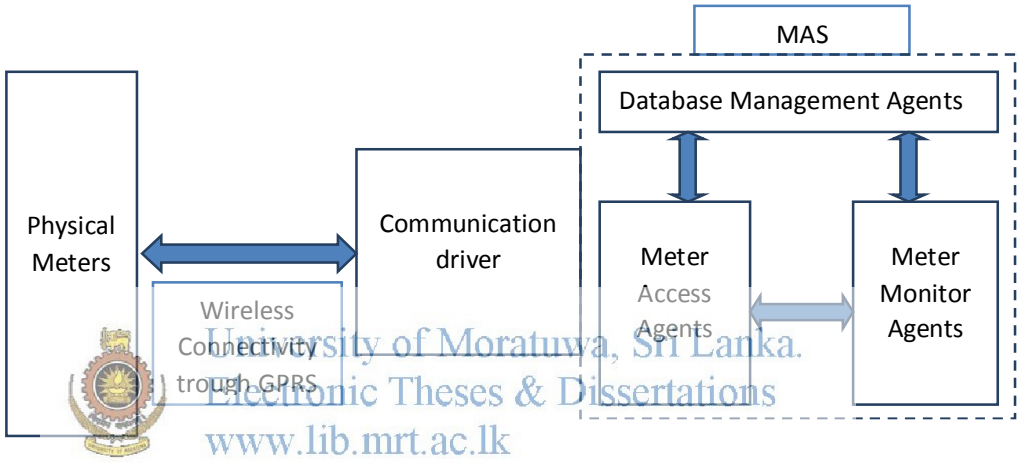


Figure 3.1: Main Components of MRS

3.2 Physical Meter Base

Current Transformer (CT) operated Poly Phase Meters (PPM) which is complying with DLMS specification is used for the implementation of the system. Hence the meters are complying with DLMS specification, those follows IEC standard for meter communication. The meters support for GPRS communication.

3.3 Communication Driver

Communication driver is developed to establish communication between MAS and physical meters through GPRS communication. Communication driver was developed based on IEC standard. IEC 62056-42(Electricity metering - Data exchange for meter reading, tariff and load control - Part 42: Physical layer services

and procedures for connection-oriented asynchronous data exchange), IEC 62056-21 (Electricity metering – Data exchange for meter reading, tariff and load control – Part 21: Direct local data exchange) and the IEC 62056-42 (Electricity metering data exchange - The DLMS/COSEM suite - Part 6-1: Object Identification System (OBIS)). Figure: 3.2 illustrate the typical physical component diagram described in IEC 62056-42. And the Figure 3.3 illustrates the component diagram of physical implementation.

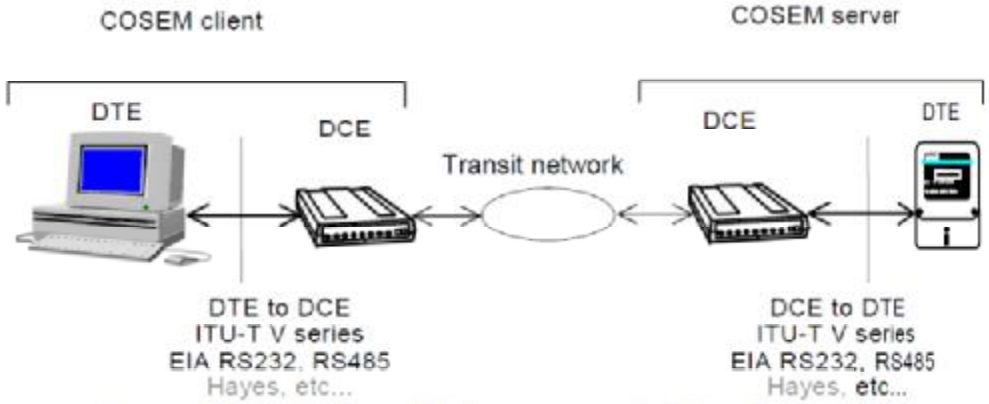


Figure 3.2 Typical PSTN configuration as in IEC 62056-42

Figure 3.3 Component Configuration Physical Implementation

3.3.1. Methodology of Communication Driver

Initially socket communication tunnel is created by communication driver. After handshaking with the meter, the meter number is sent by the communication driver

as the device address for the identification process. Hereafter meter reading command is sent to the meter and gets the required meter reading and the socket communication tunnel is closed. Figure: 3.4 and Figure: 3.5 illustrate the block diagram of communication protocol describe as in the IEC 62056-21

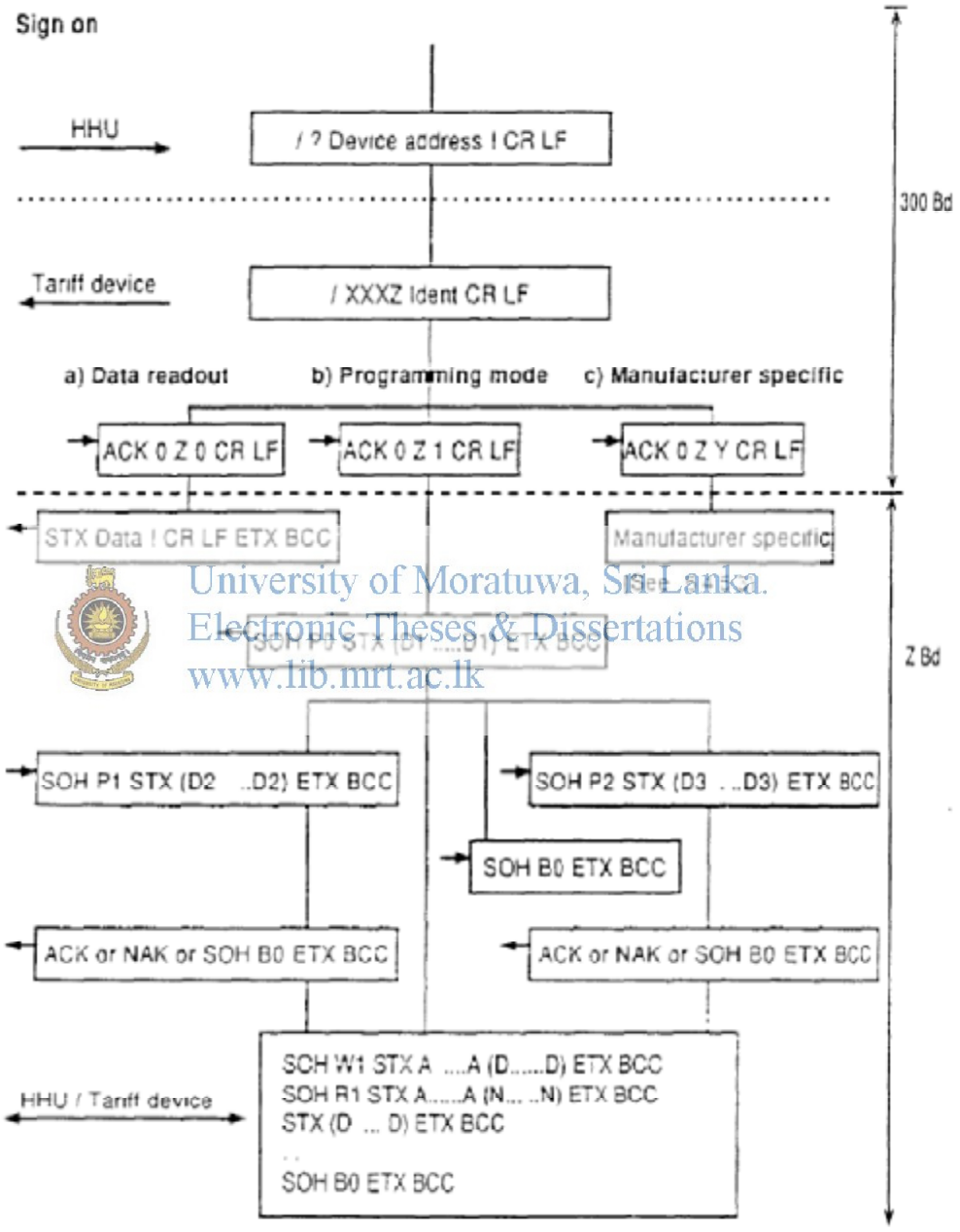


Figure 3.4: Protocol Diagram in IEC 62056-21

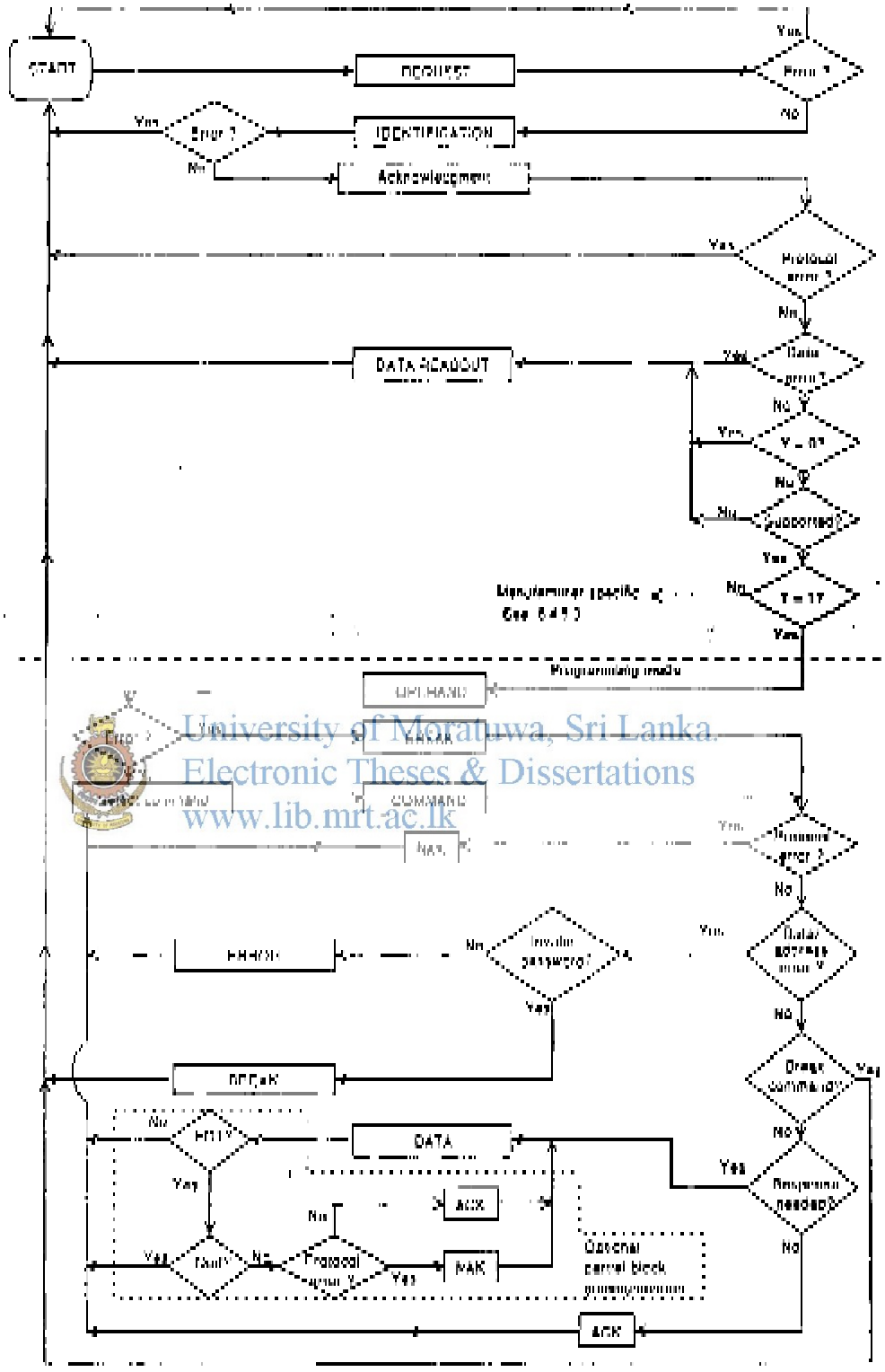


Figure 3.5: Flow chart for Direct Local Data Exchange Protocol in IEC 62056-21

The communication driver developed according to above mention protocol using JAVA programming language. The programming code given in Appendix A and Figure 3.6 illustrated the driver develop for communication.

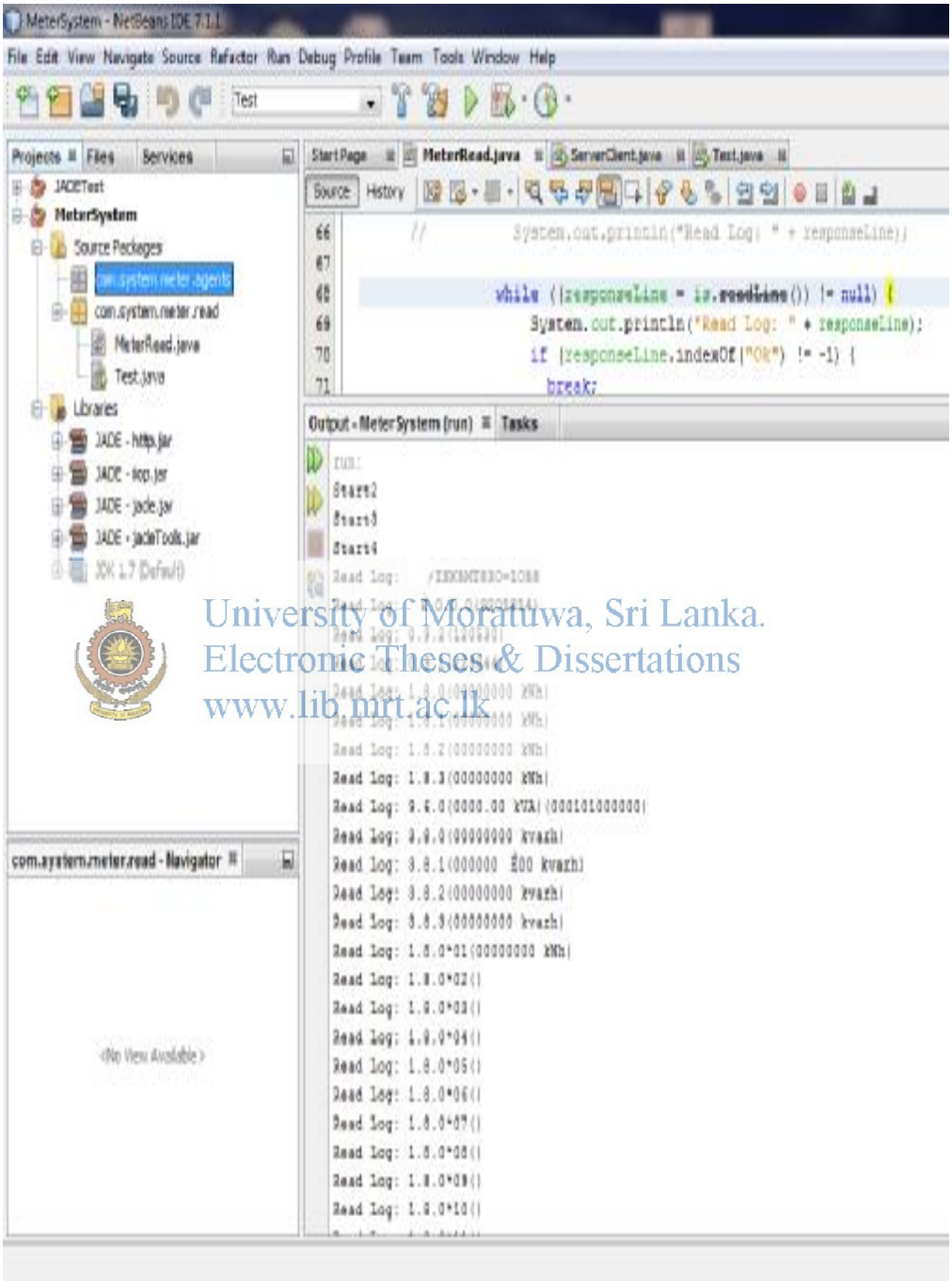


Figure 3.6: Communication Driver Interface

3.4 Multi Agent System

MAS consist of three types of Multi Agents namely Database Management Agents, Meter Access Agents and Meter Monitor Agents. In the below sections it will illustrate into the depth of the agent types used, their behaviours to achieved desired target. Each agent type has its own responsibilities and capabilities.

3.4.1. Meter monitor agents

Meter Monitor Agents is the top manager of a meter reading system. Its main responsibilities are as follows,

- Main responsibility is the generating optimum plan for Meter Access Agent.
- Listening Meter Access Agents and get the followings
    - Respond time
    - Work load Data
    - Availability
- Calculate optimum meter assignment for the Meter Access Agent based on received data
- Send the new plan for Meter Access Agents
- providing data to end applications

Main function of this agent is to generate optimum plan for the meter access agents based on data which sent by them dynamically. This process is describes in the Figure 3.7. Calculating algorithms which can be used for that purpose are listed below.

- Respond time of "X" th agent in attempt "k" = $T_{x(k)}$
- Number of jobs competed of "X" th agent = k
- New respond time of "X" th agent = $T_{x(k+1)} = \frac{\sum_{k=k-10}^{k} T_{x(k)}}{10}$
- Work load of "X" th agent= L_x
- Number of agents = n
- New work load "X" th agent $L_{x(k+1)} = \frac{(L_{new}+\sum_{x=1}^{n} L_x)\sum_{x=1}^{n} T_{x(k+1)}}{nT_{x(k+1)}\sum_{x=1}^{n} L_x}$
- Total estimated time $T = \sum T_{x(k+1)} L_{x(k+1)}$

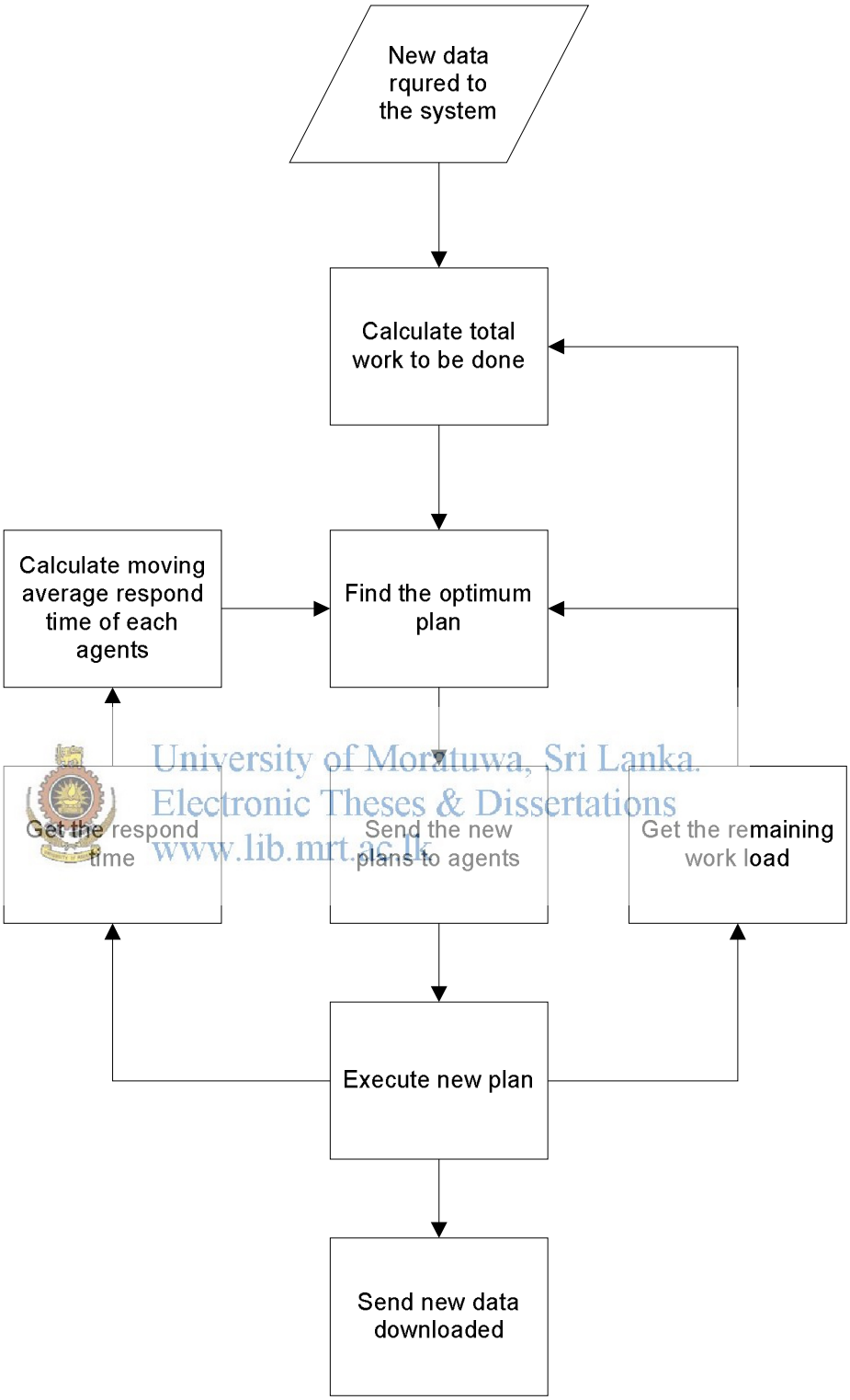


Figure 3.7: Bock Diagram of Monitor Agent Behaviour

3.4.1.1. Methodology of Meter Monitor Agents

The Meter Monitor agents have several behaviours to complete it responsibilities. Those behaviours listed below.

- Meter Assigning Behavior
  - This is used to generate meter assignment plan.
- Meter Pickup Behavior
  - This is used to get the meter information from Database Agents.
- Meter Reading Pickup Behavior
  - This is used to get the meter reading form Meter Agents.
- Notify Meter Assignment Behavior
  - This is used to assign meter to Meter Agents.
- Meter Agent Pickup Behavior
  - This is used to get the available meter agents.

The Meter Monitor Agents using JAVA programming language in JADE environment. The programming code given in Appendix B and Figure 3.8 illustrated the Interface developed for Monitor Agent.

3.4.2. Meter access agents

Meter Access Agent is the second most important agent of a meter reading system. The process of Meter Access Agents are describes in the Figure 3.9.

Its main responsibilities are as follows,

- Main responsibility is accessing meters and getting required data according to the plan send by the meter monitor agents.
- Look up for available monitors
- Inform Startup and present work load
  - Respond time
  - Work load Data
  - Availability
- download the meter reading according to the plan
- Send the collected data

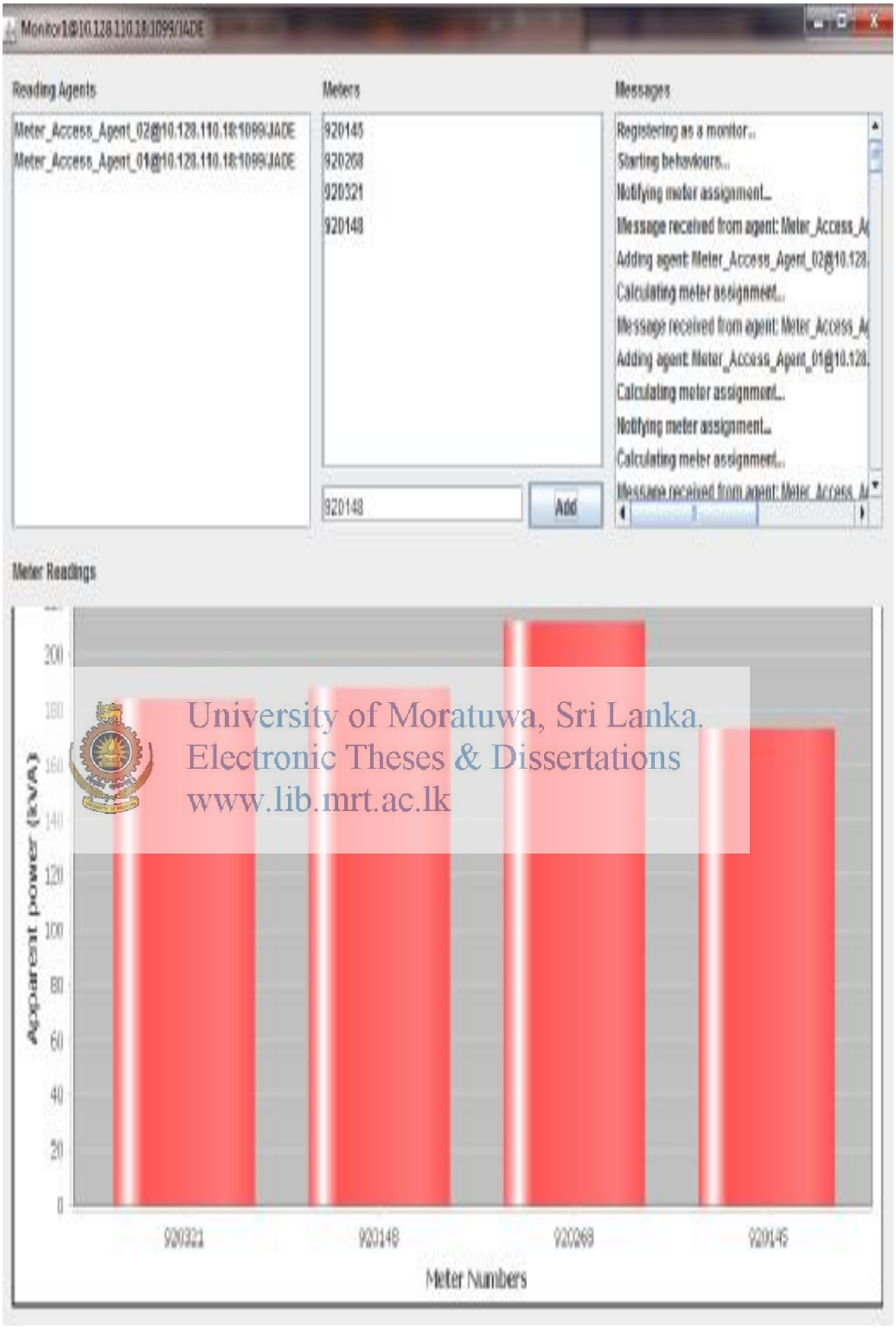


Figure 3.8: Monitor Agent Interface

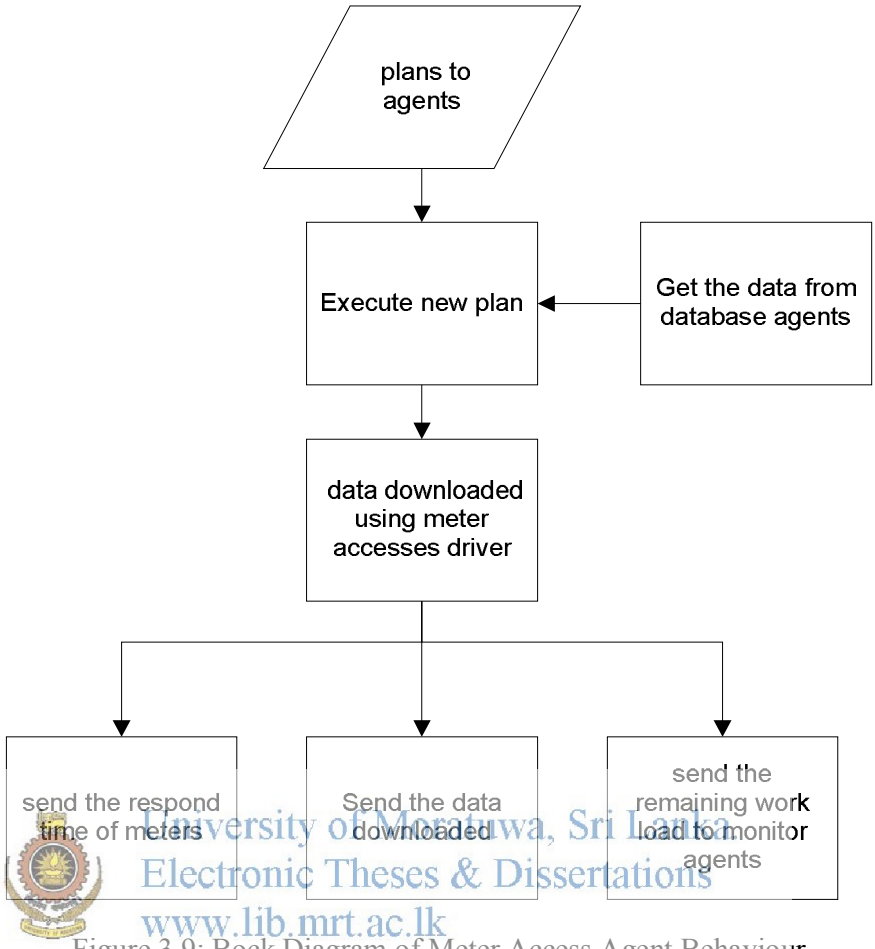


Figure 3.9: Bock Diagram of Meter Access Agent Behaviour

3.4.2.1. Methodology of Meter Access Agents

The Meter Access Agents have several behaviours to complete it responsibilities. Those behaviours listed below.

- Assignment Pickup Behavior
  - . This is used to capture meter reading plan send by the Monitor Agents
- Notify Meter Reading Behavior
  - This is used to inform downloaded data.
- Notify Shutdown Behavior
  - This is used to termination of the agent.
- Notify Startup Behavior
  - This is used to inform startup of the agent.

The Meter Access Agents using JAVA programming language in JADE environment. The programming code given in Appendix C and Figure 3.10 illustrated the Interface developed for Meter Access Agents.

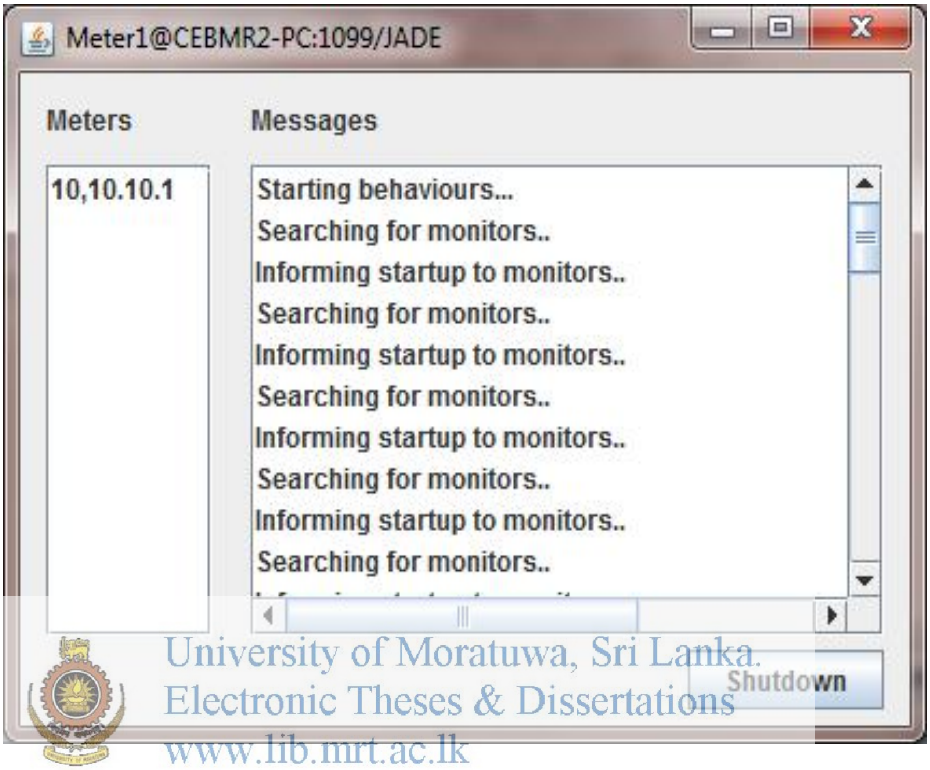


Figure 3.10: Meter Access Agent Interface

3.4.3. Database agent

Database Agent provides meter information to other agents running in meter reading system. The process of Database Agent are describes in the Figure 3.11. Its main responsibilities are as follows,

- Main responsibility is to synchronize distributed database and provide meter information to other agents running in meter system
- Look up for available monitors
- Look up for available database
- Inform startup
- Provide meter data such as serial number ,IP, etc agents running in meter system

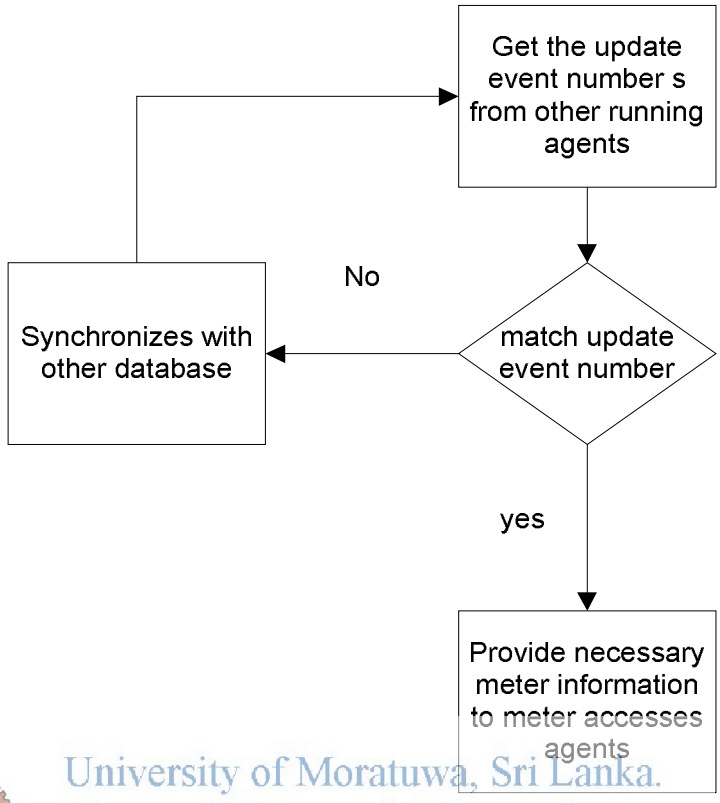


Figure 3.11: Bock Diagram of Database Agent Behaviour

3.4.3.1. Methodology of Database Agents

The Database Agents have several behaviours to complete it responsibilities. Those behaviours listed below.

- Monitor Discovery Behavior
  - . This is used to find available Monitor Agents.
- Notify Meter Reading Behavior
  - This is used to inform meter information.
- Data synchronous Behavior
  - . This is used to synchronize the database.

The Database Agents using JAVA programming language in JADE environment. The programming code given in Appendix D and Figure 3.12 illustrated the Interface developed for Database Agents.

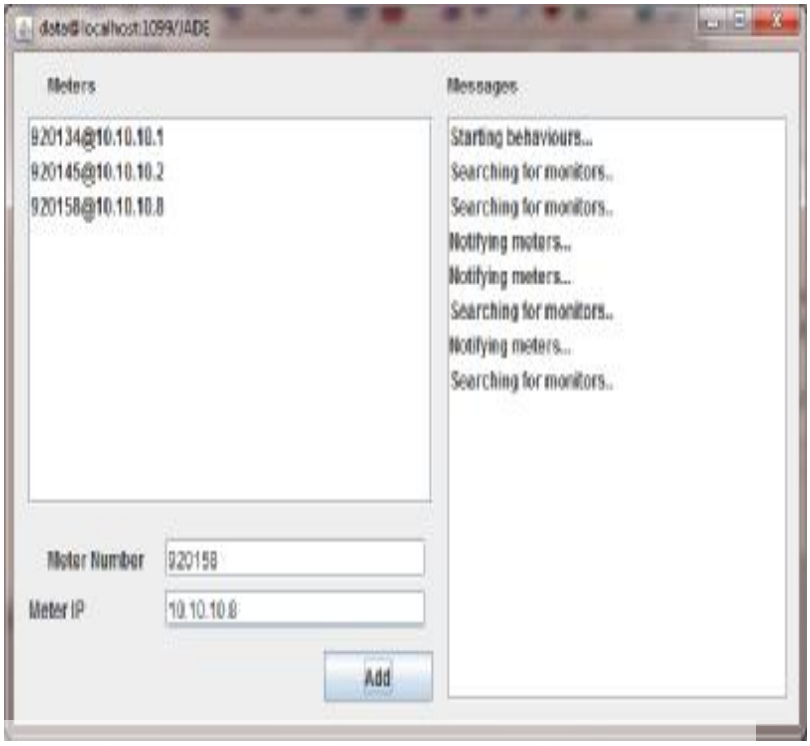


Figure 3.12: Database Agent Interface

3.5 Overall System Architecture

Meter reading system runs with the collaboration of meter monitor agent, meter access agent, data base agent and meter communication driver. Figure 3.10 illustrates overall system architecture. Meter monitor agent is the main control agent in the system. It is located the sites where meter reading monitoring is required. Meter access agents are distributed everywhere in the utility computer network. Data base agents are also distributed everywhere in the utility computer network.

When meter access agents are running, its availability is broadcasted. Then meter data downloading plans for access agents are created by meter monitoring agents and sent to the available meter access agents. According to that plan meter access agents retrieved meter accessing details, such as meter communication IP addresses, meter serial numbers etc. Then meter access agents connected to the relevant meters and

data downloaded through meter communication drivers. Then downloaded data will be sent back to meter monitoring agents.

According to the response time of meter access agents, meter accessing plans will be optimised. This system facilitates to share the duties of meter access agents in which was collapsed while they are working, among the other working agents. In other words the overall system is not affected, although several meter access agents collapsed.

As data base agents spread throughout utility network, this system itself work as redundancy system.

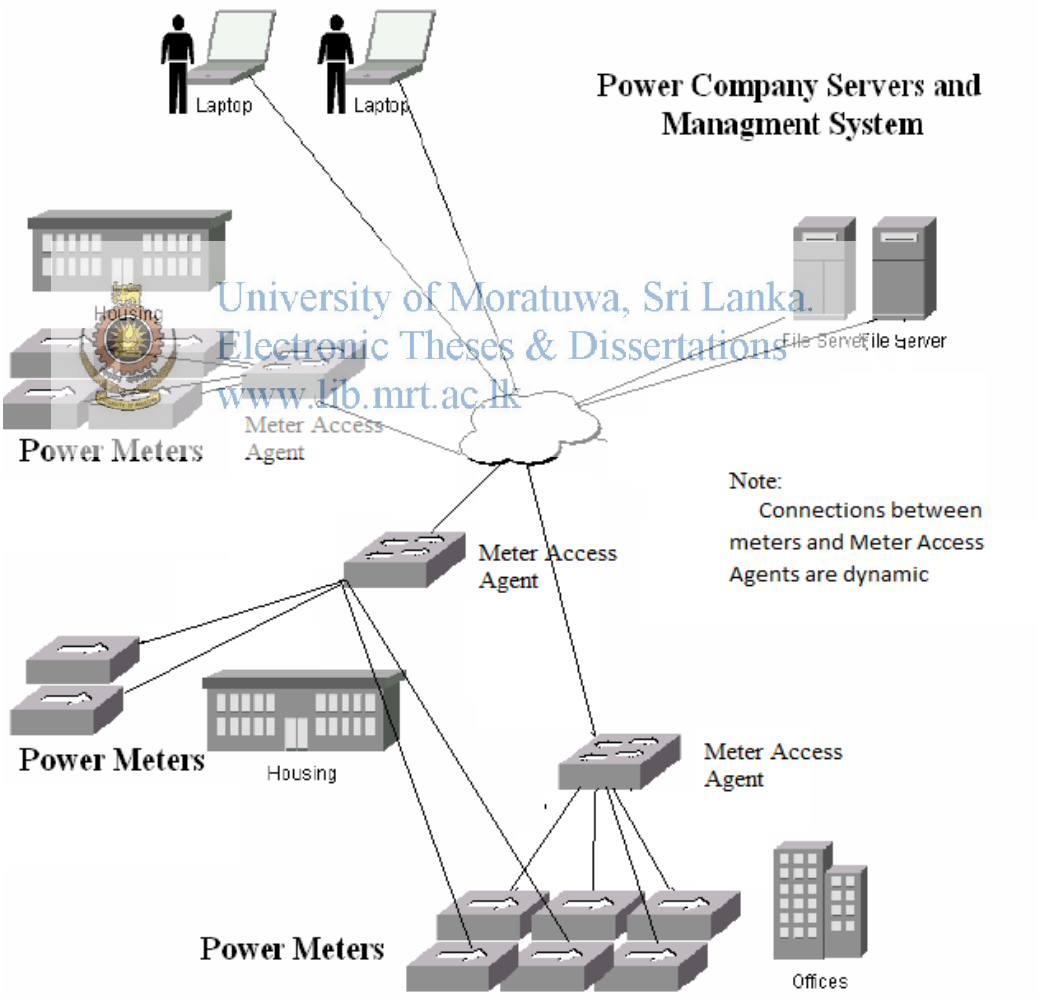
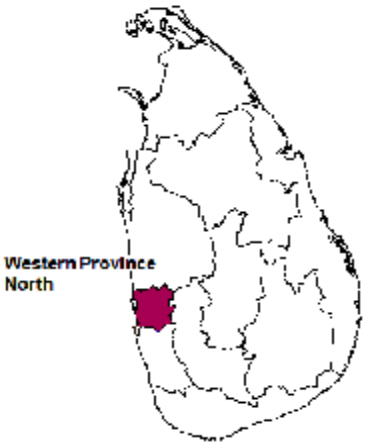


Figure 3.13: Overall Meter Reading System

4 SYSTEM IMPLEMENTATION AND RESULTS

Western Province North is divided in to 6 areas (Gampaha, Kelaniya, JaEla, Negombo, Divulapitiya and Veyangoda) for administrative purposes. Statistical data and geographic demarcation of Western Province North is mentioned below.



Geographical Demarcation	Western Province North
Electrified Houses	465,010
Land area	1421 km^2
Population	2.5 Million
Households	492,878
Bulk Consumers	1379

Figure 4.1: Statistical Data and Geographic Demarcation of Western Province North

Thousand three hundred and seventy nine bulk consumer meters were installed in that main area. Out of that six hundred and forty one DLMS specified meters are there. Number Of bulk consumer meters installed and out of that number of DLMS specified bulk consumer meters installed in each area is mentioned in the Table 4.1.

Table 4.1 . Number of Bulk Consumer Meters Installed

Area	**Number of bulk consumer meters**	**Number of DLMS specified bulk consumer meters**
Gampaha	108	63
Divlapitiya	120	69
Veyangoda	124	71
Negambo	306	230
JaEla	339	43
Kelaniya	382	165
Total	**1379**	**641**

For testing purpose the system was deployed in Negambo area. DLMS specified bulk consumer meters (230) in Negambo area were read through this system. A captured snap shot of Monitor agent when downloading meter reading is illustrated in Figure

4.2. The number of access agent in the system is varied at a time and required data downloaded and meter access delayed time measured. A captured snap shot of delayed time measurements when one meter access agent running is illustrated in Figure 4.3.

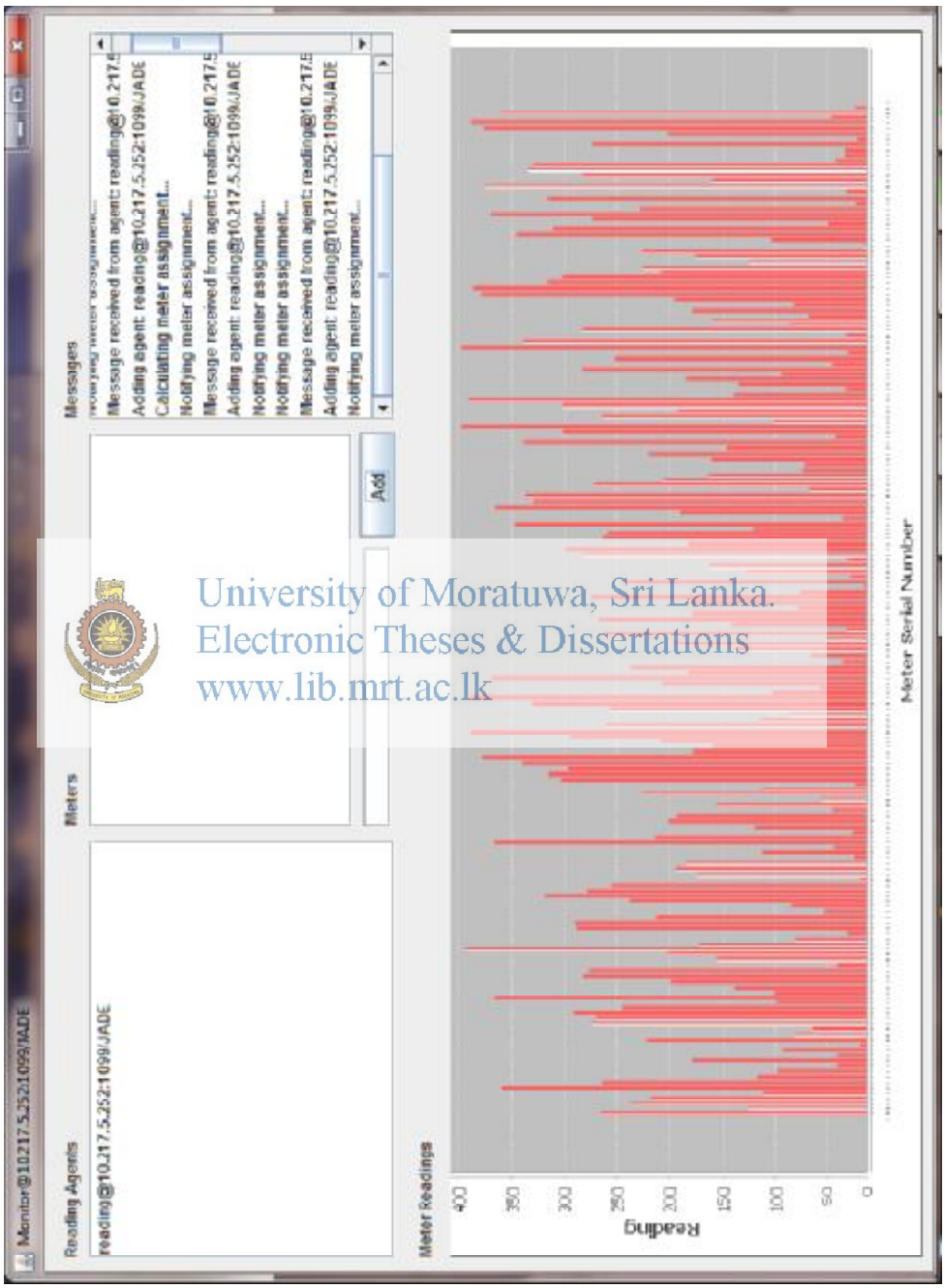


Figure 4.2: Monitor Agent when Downloading Meter Reading

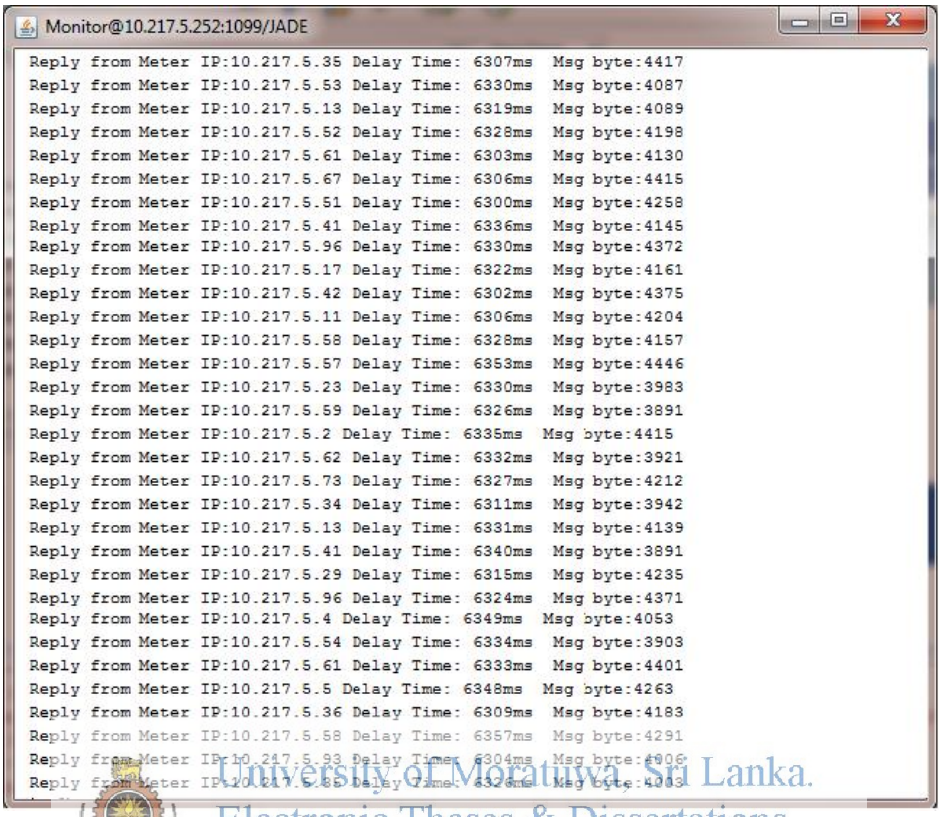


Figure 4.3: Delay Time Measurements when One Meter Access Agent Running

The number of meter access agents and the average delay time relevant to each particular number of agents mentioned in Table 4.2. Relevant captured snap shots of delayed time measurements given in Appendix E. There is a negative co-relation between the numbers of access agents and average delay time. In other words the average delay time decreases while the number of meter access agents increased.

Table 4.2: Average Delay Time

Number of Agents	Average Delay Time
1	6337
2	4718
3	3482
4	2492
5	2129
10	996
15	794
20	642

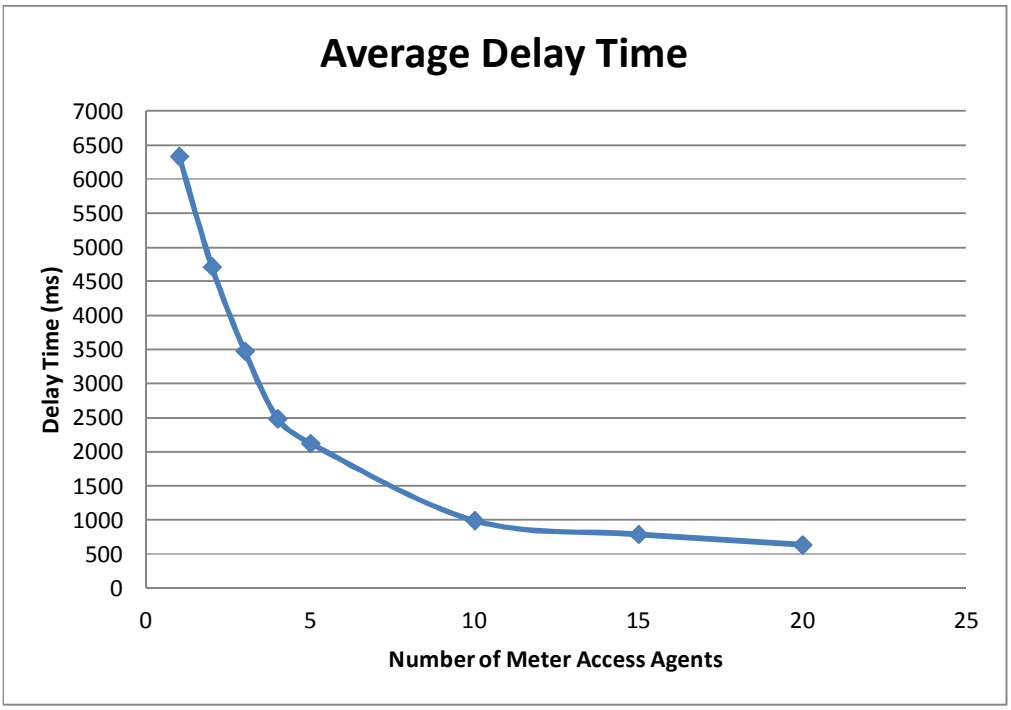


Figure 4.4: Co-relation Between the Numbers of Access Agents and Average Delay Time

5. CONCLUSION

The main objective of the MAS based implementation of real time meter reading is to provide methodological approach for energy meter reading systems to use de-centralized technique. Agent based solution was implemented with Multi Agent System (MAS) to address following issues and the system is characterized de-centralized nature and self configurable nature.

- low efficiency benefits,
- low reliability
- real time meter data monitoring get more complex when the meter base is large

JADE used as the toolkit for the agent development because JADE is at the front line, compared to the other competitive agent development toolkits. JADE provide excellent support when implementing the real time meter reading system. The management of the agent behaviours are solely controlling the agent communication and it is art of the JADE. In the multi agent system model, the agent communication is very vital for effectively implementation of complex system and interaction protocol based communication scheme and yellow pages service made the implementation easy.

The JADE agents interact with outside world via socket proxy tunnel using FIPA formatted messages. But physical equipment related to the agent communicate with outside with commonly available protocols or proprietary protocols. Therefore there should be a middle ware to match the physical world and JADE agent framework.

In this implementation, the middleware or communication layer was basically developed using the Java based application according to communication protocol mention in DLMS. Once the communication layer was developed using java, the system can interact with meters which are comply with DLMS specification.

Therefore the entire development with application layer and communication layer validate the system for the real world application.

MAS system development involved basically three type of agents namely monitor agents, meter access agent and database agents. When we concern a power distribution utility, there are many offices located everywhere in its operating region. Those agent components can be deployed into the PC that they are already been used without introducing any additional cost to the system. In other words any dedicated

high-end resources are not required to run the system. The whole task is distributed among several smaller components multi-located in network. As each component has a limited task and they are operated simultaneously, it reduces the time to carry out whole task. Hence the efficiency of the whole system get improves. According to the features of MAS, even though single component is failed, the whole system is not getting affected, because the duties of failed component are taken over by another similar working component in run time. Hence reliability of the whole system get improves. In other word no need to maintain redundancy system because the MAS itself act as redundancy system.

Multi Agent System is very flexible system and once the base operating model ensures the accuracy of the application, this can be applied for the large network. Therefore this novel concept can be used to address meter reading problem in utility network successfully.

REFERENCES

[1] G. C. D. G. Fabio Bellifemine, Developing Multi-Agent systems with JADE, West Sussex, England: Jhon Wiley & Sons Ltd, 2007.

[2] J. C. D.-w. G. J.-h. Z. Yi-Nan Guo, "A Novel Multi-agent Based Complex Process Control System and Its Application," Lecture Notes in Control and Information Sciences, Volume 344, pp. 319-330, 2006.

[3] R. Tahboub, D. Lazarescu, V. lazarescu, "Software Engineering and Database Web Management in Daleelcom Design", Proceedings of the International Conference on Information Technology: Coding and Computing(ITCC'05),IEEE,0-7695-2315-3/05, 2005.

[4] R. Tahboub, D. Lazarescu,"Secure Information Store Web Access in Time Sheet Management System", Scientific Bulletin, University Politehnica of Bucharest, S. C, V. 66, N.2-4, 2004.

[5] Tahboub.R,Lazarescu.V, "Novel Approach for Remote Energy Meter Reading Using Mobile Agents" Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on Date 10-12 April 2006

[6] Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko, and D. Rus, Mobile agents for distributed information retrieval. In Mathias Klusch, editor, Intelligent Information Agents, chapter 15, pp. 355-395, Springer-Verlag, 1999.

[7] C. Bohoris, A. Liotta, G. Pavlou, "Mobile Agent Based Performance Management for the Virtual Home Environment", Journal of Network and System Management, pp. 133-149, June 2003.

[8] Hongjun Li, "Intelligent Distributed Fault and Performance Management for Communication Networks". CSHCN PhD 2002-2.

[9] Horvat D. , Cvetkovic D. , Milutinovic V. , Kocovic P. and Kovacevic V. 'Mobile Agents and Java Mobile Agents Toolkits'. System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on 4-7 Jan. 2000

[10] Rafał Leszczyna Version 'Evaluation of Agent Platforms' Version 2.0 EUR 23508EN - 2008

[11] Nguyen G., Dang T.T., Hluchy L., Laclavik M., Balogh Z.and Budinska I., 'Agent Platform Evaluation and Comparison', Published by Institute of informatics, Slovak Academy of Sciences, Pellucid 5FP IST -2001-34519, June 2002.

[12] Vu Anh Pham,Karmouch, A. Mobile software agents: an overview

Communications Magazine, IEEE (Volume:36 , Issue: 7)

[13] *http://www.recursionsw.com/products/voyager/voyager-platforms.html [online]*

[14] *Mudumbai, Srilekha S, Johnston, William, Essiari, Abdelilah 'Anchor Toolkit - a secure mobile agent system',06-16-2008*

[15] *Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee & Jaron C. Collis 'ZEUS: A Toolkit for Building Distributed Multi-Agent Systems'*

[16] *F. Bellifemine, G. Caire, T. Trucco and G. Rimassa, JADE PROGRAMMER'S GUIDE, Free Software Foundation,, April-2010.*

Appendix A : COMMUNICATION DRIVER SOFTWARE CODE

```
package metermonitor.meterreadingagent.utilities;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.*;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 *
 * @author Dulan Ranawaka
 */
public class ReadingMeterRegister {

    Socket smtpSocket = null;
    DataOutputStream os = null;
    DataInputStream is = null;
    byte b1[] = new byte[]{3,0,8,0,0,0,0,0,0,0,0};
    byte b2[] = new byte[]{3,0,5,47,63,33,13,10};
    byte b3[] = new byte[]{3,0,6,6,48,53,48,13,10};

    String responseLine;
    public String Read(String hostname){
     try {
        smtpSocket = new Socket(hostname, 2010);
        os = new DataOutputStream(smtpSocket.getOutputStream());
        is = new DataInputStream(smtpSocket.getInputStream());
    } catch (UnknownHostException e) {
        System.err.println("Don't know about host: hostname");
    } catch (IOException e) {
        System.err.println("Couldn't get I/O for the connection to: hostname");
    }
```

```
        if (smtpSocket != null && os != null && is != null) {
            try {
                os.write(b1);
                os.write(b2);
                responseLine = is.readLine();
                os.write(b3);
                while ((responseLine = is.readLine()) != null) {
                     if( responseLine.startsWith("72.7.0")){
                          break;
                      }
                    if (responseLine.indexOf("Ok") != -1) {
                      break;
                    }
                }
                os.close();
                is.close();
                smtpSocket.close();
            } catch (UnknownHostException e) {
                System.err.println("Trying to connect to unknown host: " + e);
            } catch (IOException e) {
                System.err.println("IOException:  " + e);
            }
        }
            if (responseLine != null){
                responseLine=responseLine.split("\\(")[1].split(".")[0];
            }
            else{
                responseLine= Integer.toString(0);
            }
            return responseLine;
        }

}
```

Appendix B : MONITOR AGENT SOFTWARE CODE

```
package metermonitor.monitoragent.agent;

import jade.core.AID;
import jade.core.Agent;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import metermonitor.common.utilities.CommonConstants;
import metermonitor.monitoragent.behaviours.MeterAssigningBehaviour;
import metermonitor.monitoragent.behaviours.MeterPickupBehaviour;
import metermonitor.monitoragent.behaviours.MeterReadingPickupBehaviour;
import metermonitor.monitoragent.behaviours.NotifyMeterAssignmentBehaviour;
import metermonitor.monitoragent.behaviours.ReadingAgentPickupBehaviour;
```



```
/**
 *
 * @author Dulan Ranawaka
 */
public class MonitorAgent extends Agent {

    List<AID> _readingAgents = new ArrayList<>();
    List<String> _meters = new ArrayList<>();
    Hashtable<String, String> _meterNumbers = new Hashtable<>();
    Hashtable<String, String> _meterIps = new Hashtable<>();
    Hashtable<AID, List<String>> _meterAssignment = new Hashtable<>();
    MonitorAgentForm _agentUiForm = null;

    @Override
    protected void setup() {
        _agentUiForm = new MonitorAgentForm(this);
        _agentUiForm.setVisible(true);
        _agentUiForm.setAgentName(getAID().getName());
```

```
        showMessage("Registering as a monitor...");

        ServiceDescription serviceDescription = new ServiceDescription();
        serviceDescription.setType(CommonConstants.MonitorServiceType);
        serviceDescription.setName(CommonConstants.MonitorServiceName);

        DFAgentDescription df = new DFAgentDescription();
        df.setName(getAID());
        df.addServices(serviceDescription);

        try {
            DFService.register(this, df);
        } catch (FIPAException ex) {
            Logger.getLogger(MonitorAgent.class.getName()).log(Level.SEVERE, null,
ex);
        }

        showMessage("Starting behaviours...");

        addBehaviour(new ReadingAgentPickupBehaviour());
        addBehaviour(new NotifyMeterAssignmentBehaviour(this, 10000,
_meterAssignment));
        addBehaviour(new MeterReadingPickupBehaviour());
        addBehaviour(new MeterPickupBehaviour());
    }

    public void addReadingAgent(AID readingAgent) {
        showMessage("Adding agent: " + readingAgent.getName());

        for (AID ra : _readingAgents) {
            if (ra.getName().equals(readingAgent.getName())) return;
        }

        _readingAgents.add(readingAgent);

        addBehaviour(new MeterAssigningBehaviour(_readingAgents, _meters));

        List<String> agentNames = new ArrayList<>();
        _readingAgents.stream().forEach((ra) -> {
            agentNames.add(ra.getName());
        });
```

```
        _agentUiForm.updateReadingAgents(agentNames.toArray(new
String[agentNames.size()]));
    }

    public void removeReadingAgent(AID readingAgent) {
        showMessage("Adding agent: " + readingAgent.getName());

        AID agenttoRemove = null;

        for (AID ra : _readingAgents) {
            if(ra.getName().equals(readingAgent.getName())) {
                agenttoRemove = ra;
                break;
            }
        }

        if (agenttoRemove == null) return;
        _readingAgents.remove(agenttoRemove);

        addBehaviour(new MeterAssigningBehaviour(_readingAgents, _meters));

        List<String> agentNames = new ArrayList<>();
        _readingAgents.stream().forEach((ra) -> {
            agentNames.add(ra.getName());
        });

        _agentUiForm.updateReadingAgents(agentNames.toArray(new
String[agentNames.size()]));
    }

    public boolean addMeter(String meter) {
        if (!_meterNumbers.containsKey(meter)) return false;
        if (_meters.stream().anyMatch(m -> m.equals(meter))) return false;

        _meters.add(_meterNumbers.get(meter));

        addBehaviour(new MeterAssigningBehaviour(_readingAgents, _meters));

        return true;
    }
```

```
    public void updateMeterAssignment(Hashtable<AID, List<String>>
meterAssignment) {
        _meterAssignment.clear();
        _meterAssignment.putAll(meterAssignment);
    }

    public void addMeterInfo(String meterNumber, String meterIp) {
        if (_meterNumbers.containsKey(meterNumber) ||
_meterIps.containsKey(meterIp)) return;
        _meterNumbers.put(meterNumber, meterIp);
        _meterIps.put(meterIp, meterNumber);
    }

    public void updateMeterReading(String meter, int reading) {
        _agentUiForm.updateMeterReading(_meterIps.get(meter), reading);
    }

    public void showMessage(String message) {
        _agentUiForm.addMessage(message);
    }
}
```

```
package metermonitor.monitoragent.agent;

import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.util.Logger;
import java.awt.BorderLayout;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.Timer;
import java.util.TimerTask;
import javax.swing.DefaultListModel;
import javax.swing.ListModel;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
```



```
/**
 *
 * @author Dulan Ranawaka
 */
public class MonitorAgentForm extends javax.swing.JFrame {

    private final MonitorAgent _agent;
    private final Hashtable<String, Integer> _meterReadings = new Hashtable<>();

    /**
     * Creates new form MonitorAgentForm
     * @param agent
     */
    public MonitorAgentForm(MonitorAgent agent) {
        _agent = agent;

        initComponents();
```

```
    Timer chartUpdateTimer = new Timer();
    chartUpdateTimer.schedule(new TimerTask() {
        @Override
        public void run() {
          updateChart();
        }
    }, 3000, 3000);
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    btnAddMeter = new javax.swing.JButton();
    txtMeter = new javax.swing.JTextField();
    jScrollPane2 = new javax.swing.JScrollPane();
    lstMessages = new javax.swing.JList();
    jPanel1 = new javax.swing.JPanel();
    jLabel1 = new javax.swing.JLabel();
    jScrollPane3 = new javax.swing.JScrollPane();
    lstReadingAgents = new javax.swing.JList();
    jScrollPane1 = new javax.swing.JScrollPane();
    lstMeters = new javax.swing.JList();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    btnAddMeter.setText("Add");
    btnAddMeter.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            btnAddMeterActionPerformed(evt);
        }
    });
```

```
        jScrollPane2.setViewportView(lstMessages);

        jPanel1.setBorder(javax.swing.BorderFactory.createLineBorder(new
java.awt.Color(0, 0, 0)));

        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGap(0, 0, Short.MAX_VALUE)
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGap(0, 376, Short.MAX_VALUE)
        );

        jLabel1.setText("Reading Agents");
        jScrollPane3.setViewportView(lstReadingAgents);

        jScrollPane1.setViewportView(lstMeters);

        jLabel2.setText("Meters");

        jLabel3.setText("Messages");

        jLabel4.setText("Meter Readings");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addComponent(jScrollPane3,
javax.swing.GroupLayout.PREFERRED_SIZE, 318,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel1))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel2)
                                .addGap(226, 226, 226))
                            .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 299, Short.MAX_VALUE)))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(342, 342, 342)
                        .addComponent(txtMeter,
javax.swing.GroupLayout.PREFERRED_SIZE, 213,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(btnAddMeter,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                .addGap(12, 12, 12)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                    .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE, 295,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                    .addComponent(jLabel3)))
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()
                    .addComponent(jPanel1,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                .addContainerGap())
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel4)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                    .addComponent(jLabel1)
                    .addComponent(jLabel2)
                    .addComponent(jLabel3))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG, false)
                    .addComponent(jScrollPane3,
javax.swing.GroupLayout.DEFAULT_SIZE, 229, Short.MAX_VALUE)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addComponent(jScrollPane1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                        .addComponent(txtMeter,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(btnAddMeter)))
```

```
                .addComponent(jScrollPane2))
            .addGap(14, 14, 14)
            .addComponent(jLabel4)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    private void btnAddMeterActionPerformed(java.awt.event.ActionEvent evt) {
        if (!_agent.addMeter(txtMeter.getText())) return;

        ListModel meterListModel = lstMeters.getModel();
        if (meterListModel == null || !(meterListModel instanceof DefaultListModel))
        {
            meterListModel = new DefaultListModel<>();
            lstMeters.setModel(meterListModel);
        }

        ((DefaultListModel)meterListModel).addElement(txtMeter.getText());
    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnAddMeter;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JList lstMessages;
    private javax.swing.JList lstMeters;
    private javax.swing.JList lstReadingAgents;
```

```
    private javax.swing.JTextField txtMeter;
    // End of variables declaration

    public void setAgentName(String agentName) {
        this.setTitle(agentName);
    }

    public void updateReadingAgents(String[] readingAgents) {
        DefaultListModel<String> readingAgentListModel = new
DefaultListModel<>();

        for (String readingAgent : readingAgents) {
            readingAgentListModel.addElement(readingAgent);
        }

        lstReadingAgents.setModel(readingAgentListModel);
        updateChart();
    }

    public void updateMeterReading(String meter, int reading) {
        if (_meterReadings.containsKey(meter)) {
            _meterReadings.replace(meter, reading);
        } else {
            _meterReadings.put(meter, reading);
        }
    }

    private void updateChart() {
        DefaultCategoryDataset objDataset = new DefaultCategoryDataset();

        Iterator<Map.Entry<String, Integer>> it = _meterReadings.entrySet().iterator();

        while (it.hasNext()) {
            Map.Entry<String, Integer> entry = it.next();

            if (objDataset.getColumnKeys().contains(entry.getKey())) {
                objDataset.setValue(entry.getValue(), "X", entry.getKey());
            } else {
                objDataset.addValue(entry.getValue(), "X", entry.getKey());
            }
```

```
        }

        JFreeChart objChart = ChartFactory.createBarChart("Meter Readings", "Meter
Serial Number", "Meter Readings (kVA)", objDataset, PlotOrientation.VERTICAL,
false, true, false);

        ChartPanel CP = new ChartPanel(objChart);
        jPanel1.setLayout(new java.awt.BorderLayout());
        jPanel1.removeAll();
        jPanel1.add(CP,BorderLayout.SOUTH);
        jPanel1.validate();
    }

    public void addMessage(String message) {
        ListModel meterIdListModel = lstMessages.getModel();
        if (meterIdListModel == null || !(meterIdListModel instanceof
DefaultListModel))
        {
            meterIdListModel = new DefaultListModel<>();
            lstMessages.setModel(meterIdListModel);
        }
        if((DefaultListModel)meterIdListModel).size()>10)
        {
        ((DefaultListModel)meterIdListModel).clear();
         ((DefaultListModel)meterIdListModel).removeAllElements();
        }
        ((DefaultListModel)meterIdListModel).addElement(message);
    }
}
```

```
package metermonitor.monitoragent.behaviours;

import jade.core.AID;
import jade.core.behaviours.OneShotBehaviour;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;
import metermonitor.monitoragent.agent.MonitorAgent;

/**
 *
 * @author Dulan Ranawaka
 */
public class MeterAssigningBehaviour extends OneShotBehaviour {

    private final List<AID> _readingAgents;
    private final List<String> _meters;

    public MeterAssigningBehaviour(List<AID> readingAgents, List<String> meters)
{
        this._readingAgents = readingAgents;
        this._meters = meters;
    }

    @Override
    public void action() {
        if(myAgent instanceof MonitorAgent) {
            ((MonitorAgent)myAgent).showMessage("Calculating meter assignment...");
        }

        if (_readingAgents.size() == 0 || _meters.size() == 0) return;

        Hashtable<AID, List<String>> meterAssignment = new Hashtable<>();

        _readingAgents.stream().forEach((readingAgent) -> {
            meterAssignment.put(readingAgent, new ArrayList<>());
        });

        for (int i = 0; i < _meters.size(); i++) {
            meterAssignment.get(_readingAgents.get(i %
_readingAgents.size())).add(_meters.get(i));
```

```
        }
            if(myAgent instanceof MonitorAgent) {
           ((MonitorAgent)myAgent).updateMeterAssignment(meterAssignment);
        }
    }
}
package metermonitor.monitoragent.behaviours;

import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import metermonitor.monitoragent.agent.MonitorAgent;

/**
 *
 * @author Dulan Ranawaka
 */
public class MeterPickupBehaviour extends CyclicBehaviour {

    @Override
    public void action() {
        MessageTemplate messageTemplate =
MessageTemplate.MatchPerformative(ACLMessage.INFORM_REF);
        ACLMessage msg = myAgent.receive(messageTemplate);

        if (msg != null) {
            if(myAgent instanceof MonitorAgent) {
                ((MonitorAgent)myAgent).showMessage("Meter received from agent: " +
msg.getSender().getName());

                String[] messageContent = msg.getContent().split("@");
                if(messageContent.length == 0) return;

                ((MonitorAgent)myAgent).addMeterInfo(messageContent[0],
messageContent[1]);
            }
        }
        else {
            block();
        }
    }}
```

```
package metermonitor.monitoragent.behaviours;

import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import metermonitor.monitoragent.agent.MonitorAgent;

/**
 *
 * @author Dulan Ranawaka
 */
public class MeterReadingPickupBehaviour extends CyclicBehaviour {

   @Override
   public void action() {
      MessageTemplate messageTemplate =
MessageTemplate.MatchPerformative(ACLMessage.CFP);
      ACLMessage msg = myAgent.receive(messageTemplate);

      if (msg != null) {
         if(myAgent instanceof MonitorAgent){
            ((MonitorAgent)myAgent).showMessage("Meter reading received from
agent: " + msg.getSender().getName());

            String[] messageContent = msg.getContent().split("\\$");
            if(messageContent.length == 0) return;

            ((MonitorAgent)myAgent).updateMeterReading(messageContent[0],
Integer.parseInt(messageContent[1]));
         }
      }
      else {
         block();
      }
   }
}
```

```
package metermonitor.monitoragent.behaviours;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;
import java.util.Hashtable;
import java.util.List;
import metermonitor.monitoragent.agent.MonitorAgent;
/**
 *
 * @author Dulan Ranawaka
 */
public class NotifyMeterAssignmentBehaviour extends TickerBehaviour {

    private final Hashtable<AID, List<String>> _meterAssignment;

    public NotifyMeterAssignmentBehaviour(Agent a, long period, Hashtable<AID,
List<String>> meterAssignment) {
        super(a, period);
        this._meterAssignment = meterAssignment;
    }
    @Override
    protected void onTick() {
        if(myAgent instanceof MonitorAgent) {
            ((MonitorAgent)myAgent).showMessage("Notifying meter assignment...");
        }
            _meterAssignment.forEach((a, m) -> {
            AID readingAgent = a;
            if (m.size() > 0) {
                ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
                msg.addReceiver(readingAgent);

                String messageContent = "";
                for (String meter : m) {
                    messageContent = messageContent + meter + "$";
                }
                messageContent = messageContent.substring(0, messageContent.length() -
1);
                        msg.setContent(messageContent);
                myAgent.send(msg);
```

```
            }
        });
    }
}
package metermonitor.monitoragent.behaviours;

import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import metermonitor.common.utilities.CommonConstants;
import metermonitor.monitoragent.agent.MonitorAgent;

/**
 *
 * @author Dulan Ranawaka
 */
public class ReadingAgentPickupBehaviour extends CyclicBehaviour {

    @Override
    public void action() {
        MessageTemplate messageTemplate =
MessageTemplate.MatchPerformative(ACLMessage.INFORM);
        ACLMessage msg = myAgent.receive(messageTemplate);

        if (msg != null) {
            if(myAgent instanceof MonitorAgent) {
                ((MonitorAgent)myAgent).showMessage("Message received from agent: "
+ msg.getSender().getName());

                if(msg.getContent().equals(CommonConstants.StartupMessage))
                    ((MonitorAgent)myAgent).addReadingAgent(msg.getSender());
                else if(msg.getContent().equals(CommonConstants.ShutdownMessage))
                    ((MonitorAgent)myAgent).removeReadingAgent(msg.getSender());
            }
        }
        else {
            block();
        }
    }

}
```

Appendix C : ACCESS AGENT SOFTWARE CODE

```
package metermonitor.meterreadingagent.agent;

import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import metermonitor.common.utilities.CommonConstants;
import metermonitor.meterreadingagent.behaviours.AssignmentPickupBehaviour;
import metermonitor.meterreadingagent.behaviours.NotifyMeterReadingBehaviour;
import metermonitor.meterreadingagent.behaviours.NotifyStartupBehaviour;

/**
 *
 * @author Dulan Ranawaka
 */
public class MeterReadingAgent extends Agent {

    Hashtable<AID, List<String>> _meterReadingRequests = new Hashtable<>();
    Hashtable<String, List<AID>> _monitors = new Hashtable<>();
    MeterReadingAgentForm _agentUiForm = null;

    @Override
    protected void setup() {
        _agentUiForm = new MeterReadingAgentForm(this);
        _agentUiForm.setVisible(true);
        _agentUiForm.setAgentName(getAID().getName());

        showMessage("Starting behaviours...");

        addBehaviour(new NotifyStartupBehaviour(this, 15000));
        addBehaviour(new AssignmentPickupBehaviour(_meterReadingRequests));
        addBehaviour(new NotifyMeterReadingBehaviour(this, 5000, _monitors));
    }

    @Override
    protected void takeDown() {
        showMessage("Notifying shutdown to monitors...");

        if (_meterReadingRequests.isEmpty()) return;

        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
```

```
        Iterator<Map.Entry<AID, List<String>>> it =
_meterReadingRequests.entrySet().iterator();

        while (it.hasNext()) {
            Map.Entry<AID, List<String>> entry = it.next();

            msg.addReceiver(entry.getKey());
        }

        msg.setContent(CommonConstants.ShutdownMessage);
        send(msg);
    }

    public void updateMeterReadingRequests(AID monitor, List<String> meters) {
        showMessage("Updating meter reading requests...");

        Iterator<Map.Entry<AID, List<String>>> it =
_meterReadingRequests.entrySet().iterator();

        while (it.hasNext()) {
            Map.Entry<AID, List<String>> entry = it.next();

            if (entry.getKey().getName().equals(monitor.getName())) {
                it.remove();
            }
        }

        _meterReadingRequests.put(monitor, meters);

        _meterReadingRequests.forEach((m, mt) -> {
            _agentUiForm.updateMeters(mt);
        });

        for (String meter : meters) {
            if (_monitors.containsKey(meter)) {
                List<AID> existingMonitors = _monitors.get(meter);

                if (!existingMonitors.stream().anyMatch(m ->
m.getName().equals(monitor)))
                    existingMonitors.add(monitor);
            }
            else {
                List<AID> monitors = new ArrayList<>();
                monitors.add(monitor);

                _monitors.put(meter, monitors);
```

```
            }
        }
    }

    public void shutdown() {
        _agentUiForm.setVisible(false);
        doDelete();
    }

    public void showMessage(String message) {
        _agentUiForm.addMessage(message);
    }
}
```

```
package metermonitor.meterreadingagent.agent;

import java.util.List;
import javax.swing.DefaultListModel;
import javax.swing.ListModel;

/**
 *
 * @author Dulan Ranawaka
 */
public class MeterReadingAgentForm extends javax.swing.JFrame {

    private final MeterReadingAgent _readingAgent;

    /**
     * Creates new form MeterReadingAgentForm
     * @param readingAgent
     */
    public MeterReadingAgentForm(MeterReadingAgent readingAgent) {
        this._readingAgent = readingAgent;

        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jScrollPane3 = new javax.swing.JScrollPane();
        jList2 = new javax.swing.JList();
        jScrollPane1 = new javax.swing.JScrollPane();
        lstMeters = new javax.swing.JList();
        btnShutdown = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
        jScrollPane2 = new javax.swing.JScrollPane();
        lstMessages = new javax.swing.JList();
        jLabel2 = new javax.swing.JLabel();

        jList2.setModel(new javax.swing.AbstractListModel() {
            String[] strings = { "Item 1", "Item 2", "Item 3", "Item 4", "Item 5" };
            public int getSize() { return strings.length; }
            public Object getElementAt(int i) { return strings[i]; }
```

```
        });
        jScrollPane3.setViewportView(jList2);

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jScrollPane1.setViewportView(lstMeters);

        btnShutdown.setText("Shutdown");
        btnShutdown.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnShutdownActionPerformed(evt);
            }
        });

        jLabel1.setText("Meters");

        jScrollPane2.setViewportView(lstMessages);

        jLabel2.setText("Messages");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addGap(0, 0, Short.MAX_VALUE)
                        .addComponent(btnShutdown))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                            .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 275, Short.MAX_VALUE)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel1)
                                .addGap(0, 0, Short.MAX_VALUE)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                    .addComponent(jLabel2)
                    .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE, 294,
javax.swing.GroupLayout.PREFERRED_SIZE))))
            .addContainerGap())
        );
        layout.setVerticalGroup(
           layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
           .addGroup(layout.createSequentialGroup()
              .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                 .addComponent(jLabel1)
                 .addComponent(jLabel2))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG, false)
                 .addComponent(jScrollPane2)
                 .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 205, Short.MAX_VALUE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 66,
Short.MAX_VALUE)
              .addComponent(btnShutdown)
              .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void btnShutdownActionPerformed(java.awt.event.ActionEvent evt) {
        _readingAgent.shutdown();
    }


    // Variables declaration - do not modify
    private javax.swing.JButton btnShutdown;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JList jList2;
    private javax.swing.JScrollPane jScrollPane1;
```

```
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JList lstMessages;
    private javax.swing.JList lstMeters;
    // End of variables declaration

    public void setAgentName(String agentName) {
        this.setTitle(agentName);
    }

    public void updateMeters(List<String> meters) {
        DefaultListModel<String> meterListModel = new DefaultListModel<>();

        meters.stream().filter((meter) ->
(!meterListModel.contains(meter))).forEach((meter) -> {
            meterListModel.addElement(meter);
        });

        lstMeters.setModel(meterListModel);
    }

    public void addMessage(String message) {
        ListModel meterIdListModel = lstMessages.getModel();
        if (meterIdListModel == null || !(meterIdListModel instanceof
DefaultListModel))
        {
            meterIdListModel = new DefaultListModel<>();
            lstMessages.setModel(meterIdListModel);
        }
        if(((DefaultListModel)meterIdListModel).size()>10)
        {
        ((DefaultListModel)meterIdListModel).clear();
        ((DefaultListModel)meterIdListModel).removeAllElements();
        }
        ((DefaultListModel)meterIdListModel).addElement(message);
    }
}
```

```
package metermonitor.meterreadingagent.behaviours;

import jade.core.AID;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.util.Arrays;
import java.util.Hashtable;
import java.util.List;
import metermonitor.meterreadingagent.agent.MeterReadingAgent;

/**
 *
 * @author Dulan Ranawaka
 */
public class AssignmentPickupBehaviour extends CyclicBehaviour {

   private final Hashtable<AID, List<String>> _meterReadingRequests;

   public AssignmentPickupBehaviour(Hashtable<AID, List<String>>
meterReadingRequests) {
      this._meterReadingRequests = meterReadingRequests;
   }

   @Override
   public void action() {
      MessageTemplate messageTemplate =
MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
      ACLMessage msg = myAgent.receive(messageTemplate);

      if (msg != null) {
         if(myAgent instanceof MeterReadingAgent) {
            ((MeterReadingAgent)myAgent).showMessage("Reading request received
from agent: " + msg.getSender().getName());

            String[] meters = msg.getContent().split("\\$");
            if(meters.length == 0) return;


((MeterReadingAgent)myAgent).updateMeterReadingRequests(msg.getSender(),
Arrays.asList(meters));
         }
      }
      else {
         block();
      }
   }
```

```
}
package metermonitor.meterreadingagent.behaviours;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Random;
import metermonitor.meterreadingagent.agent.MeterReadingAgent;
import metermonitor.meterreadingagent.utilities.ReadingMeterRegister;
/**
 *
 * @author Dulan Ranawaka
 */
public class NotifyMeterReadingBehaviour extends TickerBehaviour {

   private final Hashtable<String, List<AID>> _monitors;

   public NotifyMeterReadingBehaviour(Agent a, long period, Hashtable<String,
List<AID>> monitors) {
      super(a, period);
      this._monitors = monitors;
   }
   @Override
   protected void onTick() {
      if(myAgent instanceof MeterReadingAgent && !_monitors.isEmpty()) {
         ((MeterReadingAgent)myAgent).showMessage("Informing meter
readings..");
      }

      Iterator<Map.Entry<String, List<AID>>> it = _monitors.entrySet().iterator();
          while (it.hasNext()) {
         Map.Entry<String, List<AID>> entry = it.next();
         ACLMessage msg = new ACLMessage(ACLMessage.CFP);

         for (AID monitor : entry.getValue()) {
            msg.addReceiver(monitor);
         }
         ReadingMeterRegister Meter_reder =new ReadingMeterRegister();

         msg.setContent(entry.getKey() + "$" + Meter_reder.Read(entry.getKey()));
         myAgent.send(msg);
      }
```

```
    }
}
package metermonitor.meterreadingagent.behaviours;

import jade.core.AID;
import jade.core.behaviours.OneShotBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import java.util.logging.Level;
import java.util.logging.Logger;
import metermonitor.common.utilities.CommonConstants;
import metermonitor.meterreadingagent.agent.MeterReadingAgent;
import metermonitor.meterreadingagent.utilities.ReadingAgentConstants;

/**
 *
 * @author Dulan Ranawaka
 */
public class NotifyShutdownBehaviour extends OneShotBehaviour {
 private AID[] _monitorAgents;

    @Override
    public void action() {
        ServiceDescription serviceDescription = new ServiceDescription();
        serviceDescription.setType(CommonConstants.MonitorServiceType);

        DFAgentDescription df = new DFAgentDescription();
        df.addServices(serviceDescription);

        try {
            DFAgentDescription[] monitors = DFService.search(myAgent, df);
            _monitorAgents = new AID[monitors.length];
            for (int i = 0; i < monitors.length; i++) {
                _monitorAgents[i] = monitors[i].getName();
            }
        } catch (FIPAException ex) {
            Logger.getLogger(MeterReadingAgent.class.getName()).log(Level.SEVERE,
null, ex);
        }
         ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        for (AID monitorAgent : _monitorAgents) {
            msg.addReceiver(monitorAgent);
        }
        msg.setContent(CommonConstants.ShutdownMessage);
```

```
        myAgent.send(msg);
    }
}
package metermonitor.meterreadingagent.behaviours;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import java.util.logging.Level;
import java.util.logging.Logger;
import metermonitor.common.utilities.CommonConstants;
import metermonitor.meterreadingagent.agent.MeterReadingAgent;
import metermonitor.monitoragent.agent.MonitorAgent;

/**
 *
 * @author Dulan Ranawaka
 */
public class NotifyStartupBehaviour extends TickerBehaviour {

    private AID[] _monitorAgents;

    public NotifyStartupBehaviour(Agent a, long period) {
        super(a, period);
    }

    @Override
    protected void onTick() {
        if(myAgent instanceof MeterReadingAgent) {
            ((MeterReadingAgent)myAgent).showMessage("Searching for monitors..");
        }

        ServiceDescription serviceDescription = new ServiceDescription();
        serviceDescription.setType(CommonConstants.MonitorServiceType);

        DFAgentDescription df = new DFAgentDescription();
        df.addServices(serviceDescription);

        try {
            DFAgentDescription[] monitors = DFService.search(myAgent, df);
            _monitorAgents = new AID[monitors.length];
```

```
            for (int i = 0; i < monitors.length; i++) {
                _monitorAgents[i] = monitors[i].getName();
            }
        } catch (FIPAException ex) {
            Logger.getLogger(MeterReadingAgent.class.getName()).log(Level.SEVERE,
null, ex);
        }

        if(myAgent instanceof MeterReadingAgent) {
            ((MeterReadingAgent)myAgent).showMessage("Informing startup to
monitors..");
        }

        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);

        for (AID monitorAgent : _monitorAgents) {
            msg.addReceiver(monitorAgent);
        }

        msg.setContent(CommonConstants.StartupMessage);
        myAgent.send(msg);
    }
}
```

Appendix D : DATABASE AGENT SOFTWARE CODE

```
package metermonitor.dataentryagent.agent;

import jade.core.AID;
import jade.core.Agent;
import java.util.AbstractList;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;
import metermonitor.dataentryagent.behaviours.MonitorDiscoveryBehaviour;
import metermonitor.dataentryagent.behaviours.NotifyMetersBehaviour;

/**
 *
 * @author Dulan Ranawaka
 */
public class DataEntryAgent extends Agent {

    private List<AID> _monitorAgents = new ArrayList<AID>();
    private Hashtable<String, String> _meters = new Hashtable<>();
    private DataEntryAgentForm _agentUiForm = null;
    @Override
    protected void setup() {
        _agentUiForm = new DataEntryAgentForm(this);
        _agentUiForm.setVisible(true);
        _agentUiForm.setAgentName(getAID().getName());

        showMessage("Starting behaviours...");

        addBehaviour(new MonitorDiscoveryBehaviour(this, 15000, _monitorAgents));
    }

    public boolean addMeter(String meterNumber, String meterIp) {
        if (_meters.containsKey(meterNumber) || _meters.containsValue(meterIp))
return false;

        _meters.put(meterNumber, meterIp);
        addBehaviour(new NotifyMetersBehaviour(_monitorAgents, meterNumber,
meterIp));
        return true;
    }

    public void showMessage(String message) {
        _agentUiForm.addMessage(message);
    }
```

```
}
package metermonitor.dataentryagent.agent;

import javax.swing.DefaultListModel;
import javax.swing.ListModel;

/**
 *
 * @author Dulan Ranawaka
 */
public class DataEntryAgentForm extends javax.swing.JFrame {

    private final DataEntryAgent _dataEntryAgent;

    /**
     * Creates new form DataEntryAgentForm
     * @param dataEntryAgent
     */
    public DataEntryAgentForm(DataEntryAgent dataEntryAgent) {
        this._dataEntryAgent = dataEntryAgent;

        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        txtMeterNumber = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        txtMeterIp = new javax.swing.JTextField();
        btnAdd = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        lstMessages = new javax.swing.JList();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jScrollPane2 = new javax.swing.JScrollPane();
        lstMeters = new javax.swing.JList();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

```
        jLabel1.setText("Meter Number");

        jLabel2.setText("Meter IP");

        btnAdd.setText("Add");
        btnAdd.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnAddActionPerformed(evt);
            }
        });

        jScrollPane1.setViewportView(lstMessages);

        jLabel3.setText("Meters");

        jLabel4.setText("Messages");

        jScrollPane2.setViewportView(lstMeters);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                        .addComponent(jScrollPane2,
javax.swing.GroupLayout.Alignment.TRAILING)
                        .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILI
NG, false)
                                .addGroup(layout.createSequentialGroup()
                                    .addComponent(jLabel2,
javax.swing.GroupLayout.PREFERRED_SIZE, 103,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
                                .addComponent(txtMeterIp,
javax.swing.GroupLayout.PREFERRED_SIZE, 218,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                                .addComponent(jLabel3)
                                .addGroup(layout.createSequentialGroup()
                                   .addComponent(jLabel1)
                                   .addGap(18, 18, 18)
                                   .addComponent(txtMeterNumber,
javax.swing.GroupLayout.PREFERRED_SIZE, 218,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                            .addGap(0, 0, Short.MAX_VALUE)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED))
                .addGroup(layout.createSequentialGroup()
                   .addGap(260, 260, 260)
                   .addComponent(btnAdd,
javax.swing.GroupLayout.PREFERRED_SIZE, 91,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 302,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel4))
             .addContainerGap())
      );
      layout.setVerticalGroup(
         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
             .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                .addComponent(jLabel3)
                .addComponent(jLabel4))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG, false)
                .addGroup(layout.createSequentialGroup()
                    .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE, 201,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                        .addComponent(jLabel1)
                        .addComponent(txtMeterNumber,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                        .addComponent(txtMeterIp,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel2))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addComponent(btnAdd))
                .addComponent(jScrollPane1))
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {
        if (!_dataEntryAgent.addMeter(txtMeterNumber.getText(),
txtMeterIp.getText())) return;

        ListModel meterListModel = lstMeters.getModel();
        if (meterListModel == null || !(meterListModel instanceof DefaultListModel))
        {
            meterListModel = new DefaultListModel<>();
            lstMeters.setModel(meterListModel);
        }
```

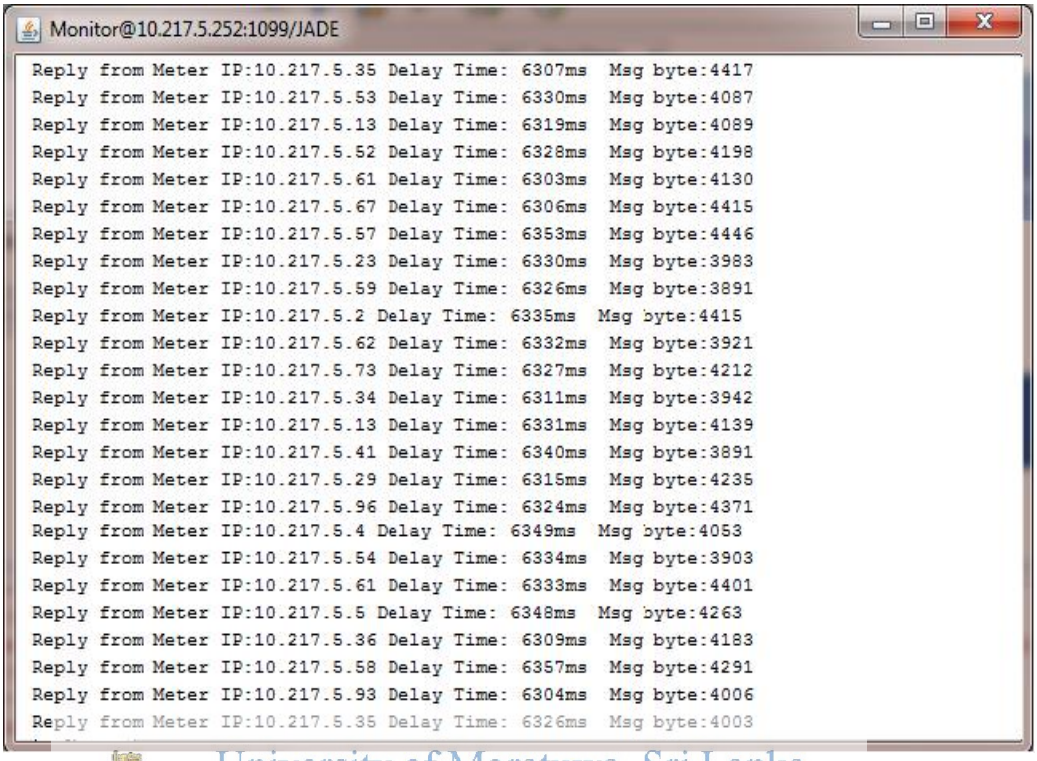
```
        ((DefaultListModel)meterListModel).addElement(String.join("@",
txtMeterNumber.getText(), txtMeterIp.getText()));

    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnAdd;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JList lstMessages;
    private javax.swing.JList lstMeters;
    private javax.swing.JTextField txtMeterIp;
    private javax.swing.JTextField txtMeterNumber;
    // End of variables declaration

    public void setAgentName(String agentName) {
        this.setTitle(agentName);
    }

    public void addMessage(String message) {
        ListModel meterIdListModel = lstMessages.getModel();
        if (meterIdListModel == null || !(meterIdListModel instanceof
DefaultListModel))
        {
            meterIdListModel = new DefaultListModel<>();
            lstMessages.setModel(meterIdListModel);
        }
        if(((DefaultListModel)meterIdListModel).size()>10)
        {
            ((DefaultListModel)meterIdListModel).clear();
            ((DefaultListModel)meterIdListModel).removeAllElements();
        }
        ((DefaultListModel)meterIdListModel).addElement(message);
    }
}
```

```
package metermonitor.dataentryagent.behaviours;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import metermonitor.common.utilities.CommonConstants;
import metermonitor.dataentryagent.agent.DataEntryAgent;
import metermonitor.meterreadingagent.agent.MeterReadingAgent;

/**
 *
 * @author Dulan Ranawaka
 */
public class MonitorDiscoveryBehaviour extends TickerBehaviour {
    private List<AID> _monitorAgents;

    public MonitorDiscoveryBehaviour(Agent a, long period, List<AID>
monitorAgents) {
        super(a, period);

        _monitorAgents = monitorAgents;
    }

    @Override
    protected void onTick() {
        if(myAgent instanceof DataEntryAgent) {
            ((DataEntryAgent)myAgent).showMessage("Searching for monitors..");
        }

        ServiceDescription serviceDescription = new ServiceDescription();
        serviceDescription.setType(CommonConstants.MonitorServiceType);

        DFAgentDescription df = new DFAgentDescription();
        df.addServices(serviceDescription);

        try {
            DFAgentDescription[] monitors = DFService.search(myAgent, df);

            _monitorAgents.clear();
```

```
            for (int i = 0; i < monitors.length; i++) {
                _monitorAgents.add(monitors[i].getName());
            }
        } catch (FIPAException ex) {
            Logger.getLogger(MeterReadingAgent.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
}
```

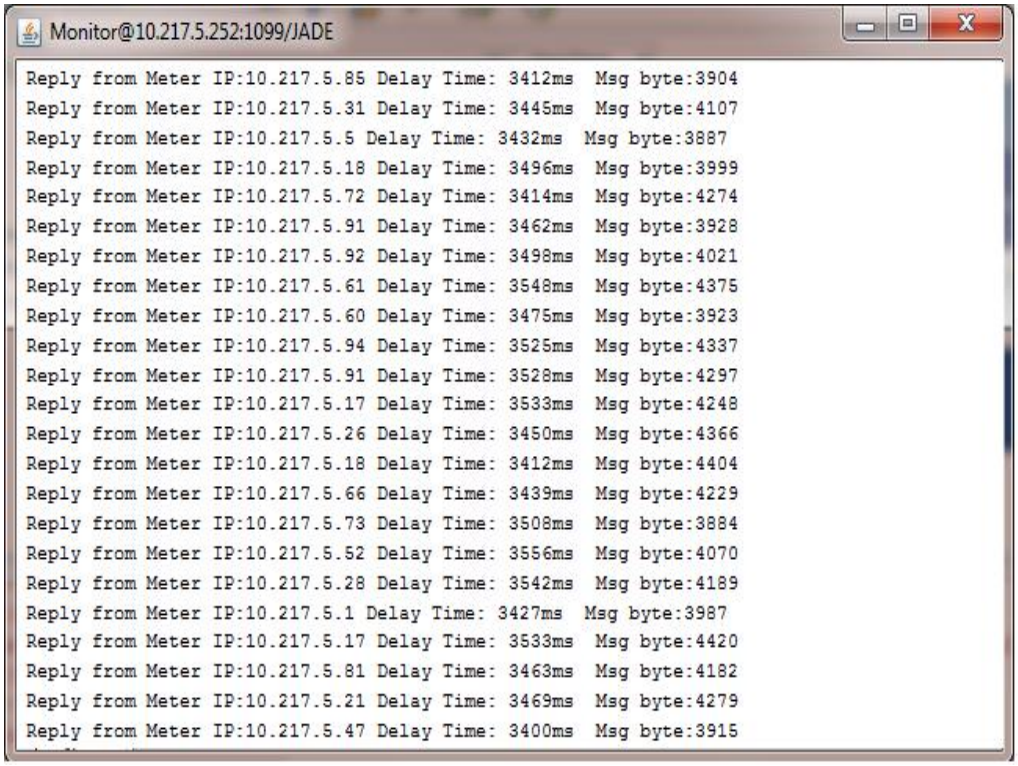
```
package metermonitor.dataentryagent.behaviours;

import jade.core.AID;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
import java.util.List;
import metermonitor.dataentryagent.agent.DataEntryAgent;

/**
 *
 * @author Dulan Ranawaka
 */
public class NotifyMetersBehaviour extends OneShotBehaviour {

    private List<AID> _monitorAgents;
    private String _meterNumber;
    private String _meterIp;

    public NotifyMetersBehaviour(List<AID> monitorAgents, String meterNumber,
String meterIp) {
        _monitorAgents = monitorAgents;
        _meterNumber = meterNumber;
        _meterIp = meterIp;
    }

    @Override
    public void action() {
        if(myAgent instanceof DataEntryAgent) {
            ((DataEntryAgent)myAgent).showMessage("Notifying meters...");
        }

        for (AID monitorAgent : _monitorAgents) {
            ACLMessage msg = new ACLMessage(ACLMessage.INFORM_REF);
            msg.addReceiver(monitorAgent);

            String messageContent = String.join("@", _meterNumber, _meterIp);
            msg.setContent(messageContent);
            myAgent.send(msg);
        }
    }

}
```

Appendix E : DELAY TIME MEASUREMENTS



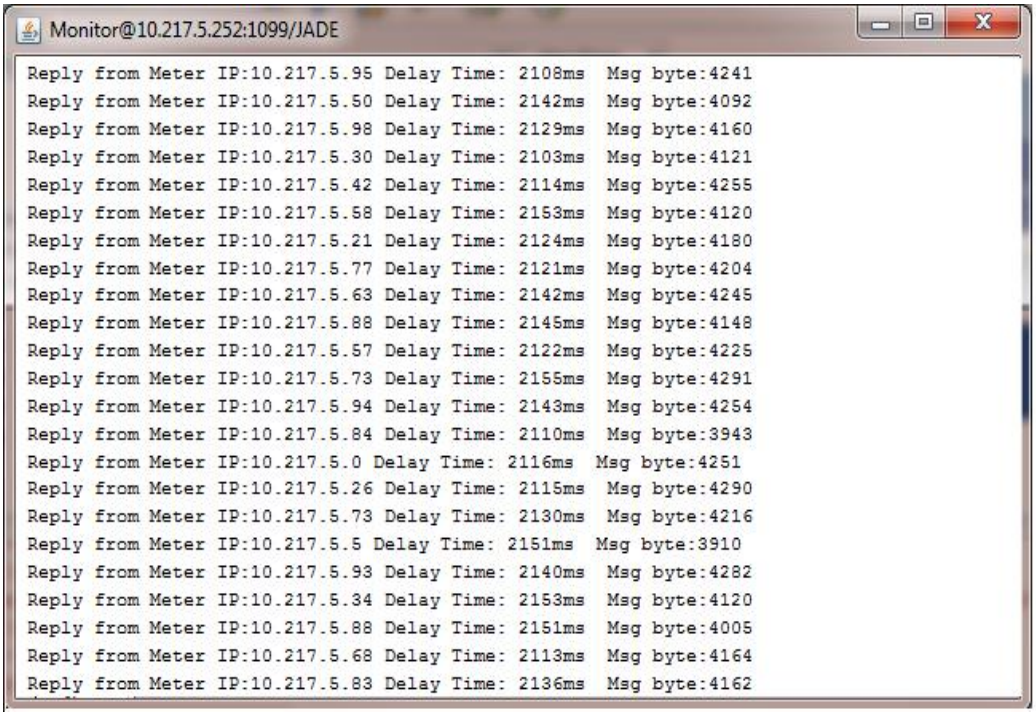
Delay Time Measurements when One Meter Access Agent Running

Delay Time Measurements when Two Meter Access Agent Running



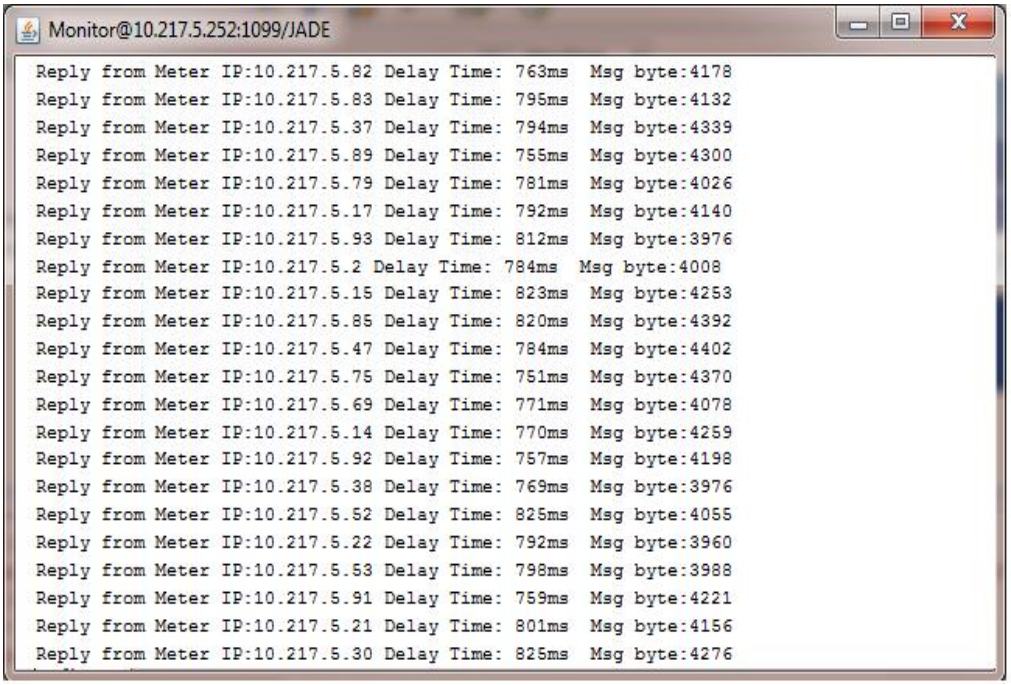
Delay Time Measurements when Three Meter Access Agent Running

Delay Time Measurements when Four Meter Access Agent Running

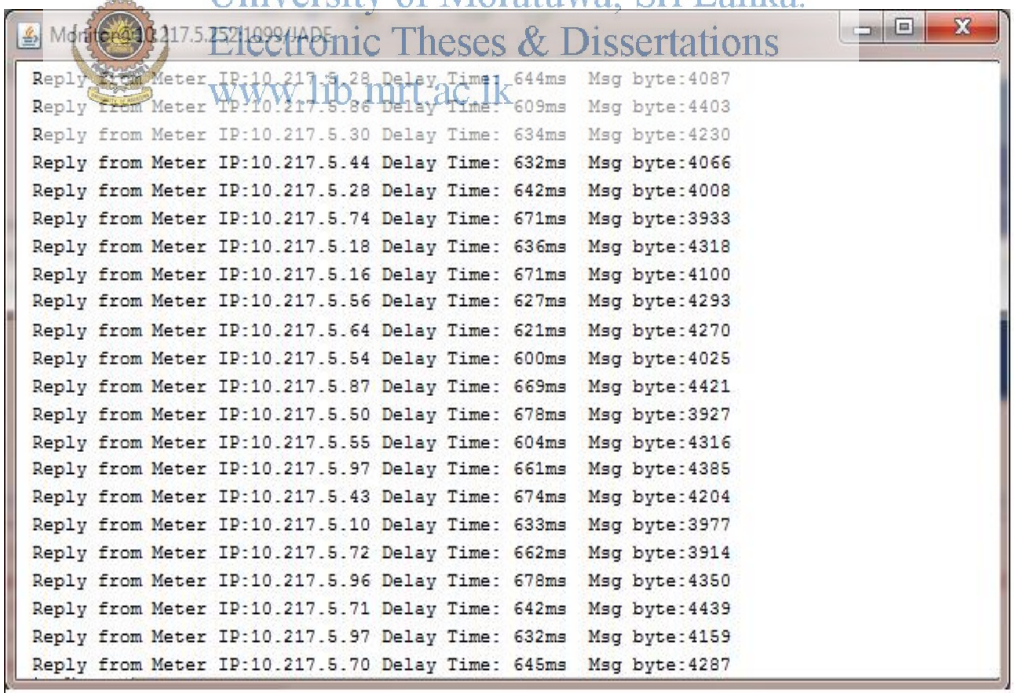


Delay Time Measurements when Five Meter Access Agent Running

Delay Time Measurements when Ten Meter Access Agent Running



Delay Time Measurements when Fifteen Meter Access Agent Running



Delay Time Measurements when Twenty Meter Access Agent Running