University of Fort Hare

Together in Excellence

A Decentralized Multi-Agent Based Network Management System for ICT4D Networks

A dissertation submitted in fulfillment of the requirements for the degree of

Masters of Science

In

Computer Science

By

Matebese Sithembiso

Supervisors: M. Thinyane and N. Moroosi

Declaration

I, Matebese Sithembiso, declare that this dissertation was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification.

Signed: _____________________

Matebese Sithembiso

Date: January 2014

Acknowledgements

Although responsibility for the final product was mine, I could not have done this work without key contributions from several individuals to whom I am deeply indebted. First, I would to thank God for all the wisdom, courage and understanding during the period of my studies. If it was not for Him, it could not have been possible. Secondly, I would like to thank the Department of Computer Science, and the Head of Department, Mr. S.M. Scott, for accepting as one of the 2012-2013 Master's Students in the first place.

To my supervisors, Professor M Thinynae and Miss N Moroosi, thanks for your technical guidance, constructive criticism and your support in all the time of this research and writing this thesis.

To my colleagues, I thank you for valuable criticism and hint you shared with me and you know without your support, I would not have got this far.

I would like also to thank Telkom SA for making this dream come through by sponsoring me through the Telkom Centre of Excellence in University of Fort Hare.

A special thanks to my family, thank you for tolerating and understanding the stress I gave you throughout my education.

Abstract

Network management is fundamental for assuring high quality services required by each user for the effective utilization of network resources. In this research, we propose the use of a decentralized, flexible and scalable Multi-Agent based system to monitor and manage rural broadband networks adaptively and efficiently. This mechanism is not novel as it has been used for high-speed, large-scale and distributed networks. This research investigates how software agents could collaborate in the process of managing rural broadband networks and developing an autonomous decentralized network management mechanism. In rural networks, network management is a challenging task because of lack of a reliable power supply, greater geographical distances, topographical barriers, and lack of technical support as well as computer repair facilities. This renders the network monitoring function complex and difficult. Since software agents are goal-driven, this research aims at developing a distributed management system that efficiently diagnoses errors on a given network and autonomously invokes effective changes to the network based on the goals defined on system agents. To make this possible, the Siyakhula Living Lab network was used as the research case study and existing network management system was reviewed and used as the basis for the proposed network management system. The proposed network management system uses JADE framework, Hyperic-Sigar API, Java networking programming and JESS scripting language to implement reasoning software agents. JADE and Java were used to develop the system agents with FIPA specifications. Hyperic-Sigar was used to collect the device information, Jpcap was used for collecting device network information and JESS for developing a rule engine for agents to reason about the device and network state. Even though the system is developed with Siyakhula Living Lab considerations, technically it can be used in any small-medium network because it is adaptable and scalable to various network infrastructure requirements. The proposed system consists of two types of agents, the *MasterAgent* and the *NodeAgent*. The *MasterAgent* resides on the device that has the agent platform and *NodeAgent* resides on devices connected to the network. The *MasterAgent* provides the network administrator with graphical and web user interfaces so that they can view network analysis and statistics. The agent platform provides agents with the executing environment and every agent, when started, is added to this platform. This system is platform independent as it has been tested on Linux, Mac and Windows platforms. The implemented system has been found to provide a suitable network management function to rural broadband networks that is: scalable in that more node agents can be added to the system to accommodate more devices in the network; autonomous in the ability to reason and execute actions based on the defined rules; fault-tolerant through being designed as a decentralized platform thereby reducing the Single Point of Failure (SPOF) in the system.

Publications

Matebese S. and Thinyane M. 2013, **Architecting a Decentralized Multi-Agent System Based Network Management System for Rural Broadband Networks (RBNs).** Proceedings of the 15th ZAWWW Conference, Cape Peninsula University of Technology, South Africa

Matebese S., Moroosi N and Thinyane M., 2013 **Implementation of Monitoring Function for MAS based Management System for RBNs.** Proceedings of SATNAC 2013 Conference, Spier Wine Estate, South Africa.

Table of Content

Table of Figures

List of Tables

Table of Acronyms

2APL: A Practical Agent Programming Language

3APL: Abstract Agent Programming Language

ACC: Agent Communication Channel

ACL: Agent Communication Language

AI: Artifical Intelligence

AID: Agent Identifier

AMAS: Adaptive Multi-Agent System

AMS: Agent Management System

API: Application Programming Interface

ARCOL: ARMITIS COmmunication Language

AWT: Abstract Window Toolkit

CIM: Common Infromation Model

CLAIM: Computational Language for Autonomous Intelligent and Mobile Agents

CLOS: Common Lisp Object System

CMIP: Communication Management information Protocol

COOL: Domain independent COOrodination Language

CORBA: Common Object Request Broker Architecture

CPU: Central Processing Unit

DF: Directory Facilitator

DNS: Domain Name System

FIPA: Foundation for Intelligent Physical Agents

FSM: Finite State Machine

GUI: Graphical User Interface

I/O: Input/Output

ICT: Information and Communication Technology

ICT4D: Information and Communication Technology for Development

IngreSQL: Ingres Structured Query Language

IP: Internet Protocol

IPMT: Internal Platform Message Transport

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

JADE: Java Agent Development Framework

Jess: Java Expert System Shell

JessML: Java Expert System Shell Markup Language

JMX: Java Management Extension

KE: Knowledge Engineering

KIF: Knowledge Interchange Format

KQML: Knowledge Query Manipulation Language

LHS: Left Hand Side

MAC: Media Access Control

MAS: Multi-Agent System

MB: Megabytes

MIB: Management Information Base

MS SQL: Microsoft Structured Query Language

MTP: Message Transport Protocol

MVEL: MVFLEX Expression language

MySQL:	My Structured Query Language
NETCONF:	Network Configuration Protocol
NIC:	Network Interface Card
NMAP:	Network Mapper
NMS:	Network Management System
OOM:	Object Oriented Methodology
OS:	Operating System
PHP:	Hypertext Preprocessor
PL/SQL:	Procedural Language/Structural Query Language
QoS:	Quality of Service
RAM:	Random-Access Memory
RMA:	Remote Monitoring Agent
RMI:	remote Method Invocation
RRD:	Round Robin Database
RRDtool:	Round Robin Database tool
SL:	Semantic Language

SLL: Siyakhula Living Lab

SNMP: Simple Network Management Protocol

SNMPc: simple Network Managment Protocol console

SNMPv3: Simple Network Management Protocol version 3

SQLite: Structured Query Lite

TCP/IP: Transimission Control Protocol/Internet Protocol

TCP: Transmission Control Protocol

TL1: Transaction Language 1

UDP: User Datagram Protocol

UML: Unified Modeling Language

UNIX: Uniplexed Operating and Computing System

URL: Uniform Resource Locator

VoIP: Voice over Internet Protocol

VSAT: Very Small Aperture Terminal

WAN: Wide Area Network

WBEM: Web-Based Enteprise Management

WiMAX: Worldwide Interoperability for Microwave Access

XML: Extensible Markup Language

1. Introduction

This chapter introduces the research scope, its significance, objectives and the motivation behind this work. It presents the research background on software agents and network management. Further, the chapter introduces the premise of this research and concludes with an outline of the dissertation structure.

1.1. Background

Telecommunication networks have emerged as an infrastructure of great significance in both developed and developing countries. As the amount of broadband network service users in rural areas significantly increases as well as the complexity of their services, network operators need to ensure high Quality of Service (QoS) required by each user. Broadband networks carry integrated traffic consisting, but not limited to voice, video and data. Rural communities are becoming more electronically oriented every year, largely due to a growth in the use of e-services (e.g. e-commerce, m-commerce and social networking technologies) in such communities. Electronic communication is becoming increasingly significant in rural areas as there is a large number of technologies being deployed for Information and Communication Technologies for Development (ICT4D) [1] [2]. Personal computing facilitates easy access, manipulation, storage and exchange of information. These processes require reliable transmission of data information between client and server for an efficient packet delivery without re-transmissions. This can be achieved by reducing network failures in the network, thereby reducing traffic congestions.

In order to achieve the goal of high QoS and efficient network provisioning, there is a need for a real time and efficient network management mechanism. Intelligent network management would play a significant role in ensuring that people in rural areas get high QoS with no interruptions. In rural networks, network management is a demanding process because of: lack of reliable power supply; greater geographical distances; topographical barriers; relatively low population density thereby reducing economies of scale; lack of technical support and computer repair facilities; and complex operational environment due to multiple role-players in ICT infrastructure implementation in these communities [3]. This makes the complexity of network monitoring function to be extremely high [4]. As a result, these networks are susceptible to network congestion resulting in delays, poor performance and inability to react without delays. For this reason, in the analysis of the exponential growth in size, distribution and complexity of communication networks, existing management mechanisms present an opportunity for improvement as far as network performance; scalability and flexibility are concerned [5].

This research is undertaken within the Siyakhula Living Lab (SLL), an ICT4D intervention that consist of 17 schools that are located in the Dwesa rural area of the Mbashe Municipality which is in the Eastern Cape province, as the research field-site.

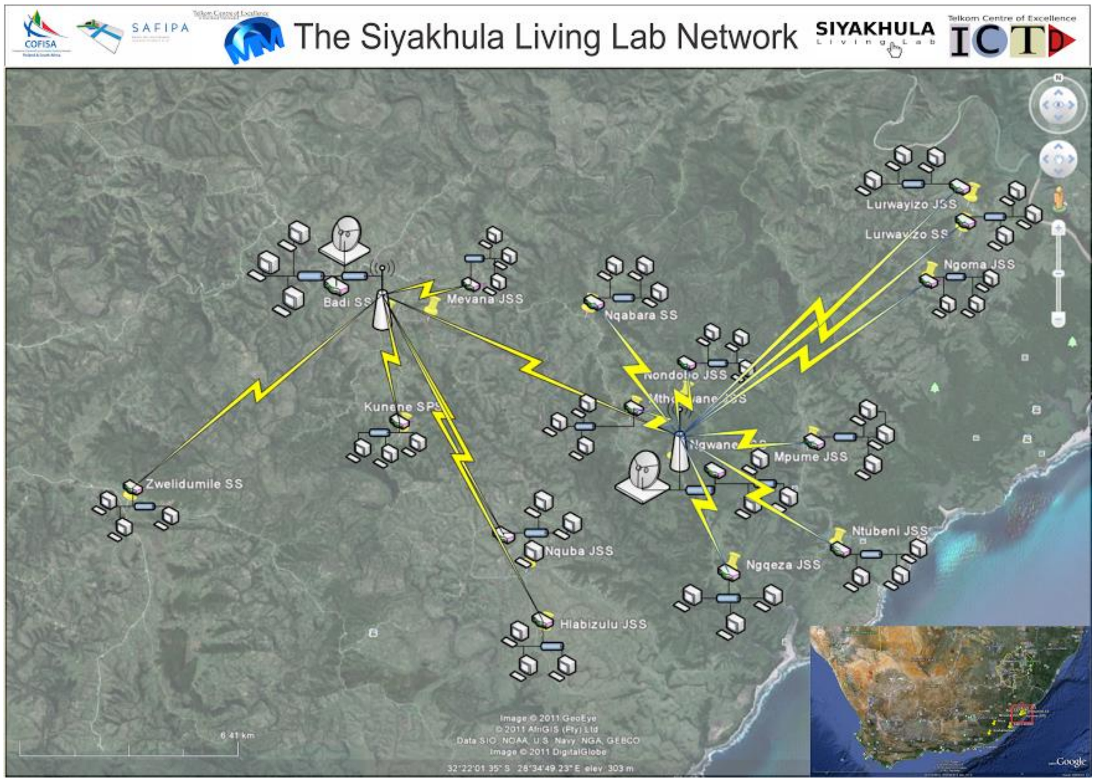


Figure 1: Geographical Network of SLL.

Figure 1 shows the geographical expanded network of the SLL situated in Dwesa rural area. Dwesa area is characteristic and typical of marginalized rural areas in South Africa: it is faced with many challenges including lack of reliable power supply, poor road conditions, telecommunication infrastructure and socio-economic challenges such as poverty and poor development. The villages targeted by the SLL are the ones surrounding the schools. The aim of the SLL is to bring ICT services to Dwesa community so that they can have access to services such as e-mail, the Internet, Voice over Internet Protocol (VoIP), e-commerce and Teleweaver multi-service platform.

1.2. Network Management

Network management encompasses the execution of a set of tasks required for access control, network planning; resource allocation, deploying, coordinating and monitoring network resources [6]. Network management tasks include: security, configuration, reliability, accounting, performance management and network inventory maintenance. These tasks are often automated during the course of monitoring and reporting services. Security management refers to protection of a network from unauthorized use; this includes external and internal unauthorized use. Security management is concerned with the right of entry to network nodes and sensitive data through using devices such as passwords. This type of management also controls the form of sensitive data using techniques such as encryption [7]. Configuration management refers to the management of security features in a network by controlling changes made to the software, hardware, firmware, documentation and test features in a system. These changes may be deliberate and may relate to the addition of a new server to the network, or related path, such as a fiber cut between two nodes resulting in a re-routed path [7]. The process of configuration management involves identifying network components and their connections, collecting each device's configuration information and defining the relationship between network components. In order to perform these tasks, the network manager needs the topological information about the network, device configuration information, and control of the network component. Accounting management is more concerned with the collection of resource consumptions data for purposes of billing, auditing, cost allocation, capacity and trend analysis [8]. This type of management information helps network administrators to give out the precise kind of resources to users, as well as plan for network growth.

Network reliability is about making sure that network resources are available to the users and responding to any malfunctions. An ideal reliable network is one that is able to quickly identify an error or failure and help initiate a quick recovery process before users experience service degradation [9]. For this, a network management system has to have three qualities: first to identify the fault, isolate the cause of the fault, and then, if possible correct the fault [9]. Performance management is about making sure that there are no bottlenecks in a network. This type of management involves measuring network resources in terms of throughput, error rates, response times and network utilization [7]. This helps network administrators to reduce or prevent network bottlenecks and/or traffic jam and also helps in providing a high quality of service to users on the network, without straining the capacity of devices and links. This type of management looks at the proportion of usage of devices and error rates to assist in improving and balancing the throughput of traffic in all components of a network. Typically, some devices are used more than others. Performance monitoring gives qualitative and up-to-date information on the health and performance of devices. This facilitates for the full utilization of underutilized devices and rebalancing of over utilized devices. In a well-consumed network with healthy mechanisms, the losses of packets on the network are few and the response times are reduced [7].

Network Inventory Management is a process that allows network administrators to retain current records about the number, type and status of network nodes [10]. An ideal process for network inventory management is one that collects inventory data on network infrastructure, regardless of vendor or technology, in one database, in which data can be updated automatically from the network. This type of management helps network administrators to collect user’s information such as the type of processes being run and establishing if the use of the devices is for legitimate reasons. This type of management can also be used to monitor external devices for threats and unauthorized content.

Network management consists of two steps, monitoring and management. Network monitoring refers to the practice of constantly gathering, storing data and reporting any faults to a network operator in the network state. Network monitoring also includes optimizing data flow and access in a complex and dynamic environment. Network management is all about reacting to any faults reported to Network Management System (NMS). Management functions centers on maintaining the network working efficiently at all times. Mainly, this requires network monitoring and examining its information for indication of possible problems.

There are protocols, standards and technologies that are used for network monitoring and management, which include: Simple Network Management Protocol (SNMP), Communication Management Information Protocol (CMIP), Web-Based Enterprise Management (WBEM), Common Information Model (CIM), Java Management Extension (JMX), Transaction Language 1 (TL1), and Network Configuration Protocol (NETCONF) among others. And these protocols and technologies can be classified as the following management mechanisms:

1. Static centralized monitoring mechanism,
2. Static decentralized monitoring mechanism,
3. Decentralized monitoring mechanism.

In recent years there has been a focus on the use of Artificial Intelligence (AI) in the management of networks, and in these instances the autonomy, collaborative operation, and robustness characteristics of intelligent agents has been leveraged to provide increased efficiency in the management of networks. This is the domain of network management that forms the basis of this research and is further discussed and elaborated in later sections.

1.3. Premise of Research

This research is conceptualized in the context of rural network management, specifically the SLL in Dwesa. Technically the deployment context of the resultant network management system could be any small to medium size network. Dwesa has poor telecommunication coverage like most other rural areas in South Africa. The network management system used in SLL is the client/server model. Figure 4 depicts the logical network diagram at SLL. The SLL network consists of two base stations located at Badi School and Ngwane School that peer together for communication and act as a redundant link to the Internet. Ngwane School uses mobile WiMAX 802.16e technology and Badi School uses fixed WiMAX 802.16d technology. These schools have the core router that is accountable for routing traffic from the connected schools to one another or the Internet; also each school has a Very Small Aperture Terminal (VSAT) connection to the Internet. Each school is equipped with a computer lab that has 5-30 thin clients running EduBuntu Linux and a few running Windows operating system.

Accurate data on the growth of Internet penetration in rural areas is hard to find, but studies around South Africa, show that by the end of 2011, 8.5 million people had access to the Internet [11]. However, out of that, 7.9 million people accessed the Internet via mobile phones. According to World Wide Worx, who conducted this research, by the end of 2013, Internet growth will be around 20% and this will result in the falling of data prices [11]. With these figures, it makes ICT4D in rural areas to be extremely challenging. One of the main objectives of SLL is to offer ICT services to the community of Dwesa. To bring socio-economic development to the community of Dwesa, SLL has to provide reliable services through the network infrastructure that is deployed. Figure 2 shows the logical network diagram currently installed in SLL [12].

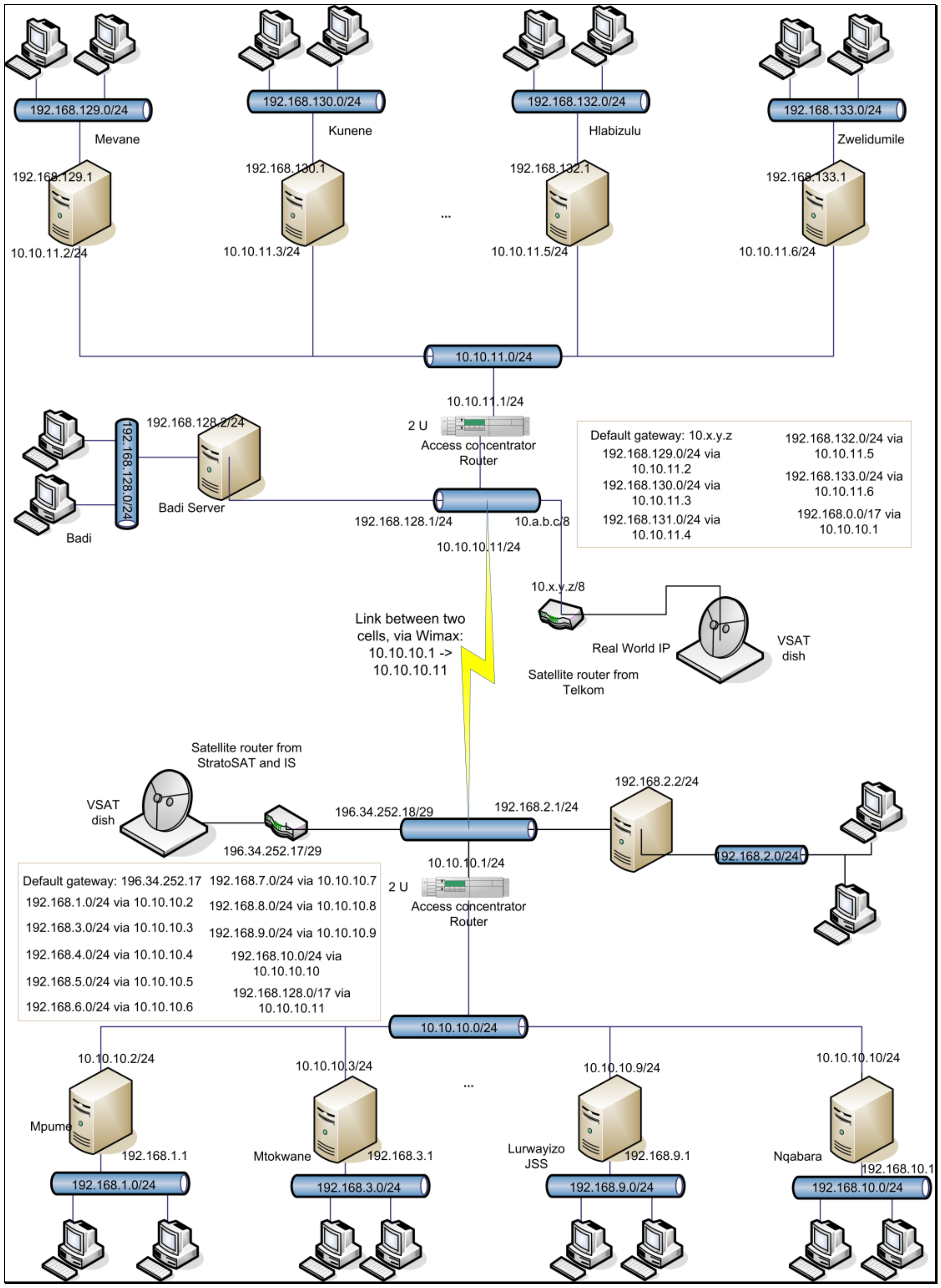

192.168.129.0/24
192.168.130.0/24
192.168.132.0/24
192.168.133.0/24
Mevane
Kunene
Hlabizulu
Zwelidumile
192.168.129.1
192.168.130.1
192.168.132.1
192.168.133.1
...
10.10.11.2/24
10.10.11.3/24
10.10.11.5/24
10.10.11.6/24
10.10.11.0/24
10.10.11.1/24
2 U
Access concentrator Router
192.168.128.2/24
192.168.128.0/24
Badi Server
Badi
192.168.128.1/24
10.a.b.c/8
10.10.10.11/24
Default gateway: 10.x.y.z
192.168.129.0/24 via 10.10.11.2
192.168.130.0/24 via 10.10.11.3
192.168.131.0/24 via 10.10.11.4
192.168.132.0/24 via 10.10.11.5
192.168.133.0/24 via 10.10.11.6
192.168.0.0/17 via 10.10.10.1
10.x.y.z/8
Link between two cells, via Wimax: 10.10.10.1 -> 10.10.10.11
Real World IP
VSAT dish
Satellite router from Telkom
Satellite router from StratoSAT and IS
VSAT dish
196.34.252.18/29
196.34.252.17/29
192.168.2.1/24
192.168.2.2/24
192.168.2.0/24
10.10.10.1/24
2 U
Access concentrator Router
Default gateway: 196.34.252.17
192.168.1.0/24 via 10.10.10.2
192.168.3.0/24 via 10.10.10.3
192.168.4.0/24 via 10.10.10.4
192.168.5.0/24 via 10.10.10.5
192.168.6.0/24 via 10.10.10.6
192.168.7.0/24 via 10.10.10.7
192.168.8.0/24 via 10.10.10.8
192.168.9.0/24 via 10.10.10.9
192.168.10.0/24 via 10.10.10.10
192.168.128.0/17 via 10.10.10.11
10.10.10.0/24
10.10.10.2/24
10.10.10.3/24
...
10.10.10.9/24
10.10.10.10/24
Mpume
192.168.1.1
Mtokwane
192.168.3.1
Lurwayizo JSS
192.168.9.1
Nqabara
192.168.10.1
192.168.1.0/24
192.168.3.0/24
192.168.9.0/24
192.168.10.0/24

Figure 2: Logical Network Diagram in SLL [12]**.**

1.4. Research Problem

The existing network management mechanisms are generally performing well, however there are a lot of challenges faced by these systems and also an opportunity for improvement. As the infrastructure of rural networks scales up, managing network resources becomes extremely difficult due to the factors mentioned in Section 1.1. This results in an increased traffic and service degradation because of increased bandwidth demand on the network. Also the requirement of human intervention and interpretation of system events demands a regular presence of a network administrator, something that is not always possible in marginalized rural settings. This can be mitigated through the use of intelligent and autonomous systems for managing these networks. Rural networks also require increase levels of robustness and fault-tolerance, which the current client-server, centralized network management systems is not always able to provide. The Single Point of Failure in these systems can be eliminated through the decentralization of the network management nodes and through the use of an intrinsically distributed and fault-tolerant platform such as is provided by Multi-Agent Systems.

1.5. Research Questions

The challenge of how to develop a multi-agent based network management system for rural broadband networks will be scrutinized thoroughly. The following questions will be addressed:

- Can an intelligent system be able to monitor and manage rural broadband networks?
- What type of agent system architecture will be suitable to manage rural broadband networks?
- What type of a Rule Engine will be able to reason with the rural broadband networks and its challenges?

The research questions above highlight the research agenda of this project.

1.6. Motivation

The proposed research of multi-agents in the application of rural network management is motivated by many factors. As the number of Internet users increases yearly, there is a need of managing available network resources effectively. Managing a network requires monitoring each node in a network. This challenge is the key force motivating this research on software agents because they can operate in environments that are dynamic, open and scalable, such as rural networks. Agents exhibit the intelligent capabilities such as reasoning, communication and learning. Agents can work collaboratively to achieve a certain task; they are goal-driven and are adaptive. Because of these properties, the use of software agents to manage rural networks would meet network management challenges (discussed in Section 1.1) of rural areas and provide dynamic, flexible, fast error detection and scalable management system.

1.7. Technical Objectives

Having introduced the problem statement of the proposed research and discussed the potential advantages of software agents in network management, the following are the specific objectives of this research:

1. Investigate how agents can be used to manage rural networks;
2. Determine the suitable agent based network architecture;
3. Develop an autonomous decentralized network management mechanism; and
4. Test if the system functions as desired to be.

1.8. Dissertation organization

With the idea of a network management system based on software agents for SLL proposed, the remaining chapters of this dissertation will give a detailed outline on how the proposed system could be achieved. In **Chapter 2**, the technology used; related work on multi-agents for network management and the case study area will be reviewed. **Chapter 3** explains how the objectives of this research are to be achieved, through an explanation of how the technologies will be combined to achieve the implementation of the system. **Chapter 4** reviews the implementation of the system. An outline of how the system will be tested to establish whether or not it is suitable for use in SLL will be presented in **Chapter 5**. **Chapter 6** offers a discussion on the findings to the study, observations and discuss about tentative future directions regarding this field.

1.9. Conclusion

The nature of network management, in particular in rural areas, is increasingly presenting challenges and opportunities that require a rethinking of the current network management operations and network management systems design. Further, it must be acknowledged that the decentralized approaches were built for either large-scale networks, high-speed networks or distributed networks without concerns about the challenges faced by rural networks in mind. Therefore, in this research, a dynamic decentralized management mechanism will be proposed. In this mechanism, management functions will be introduced at a node level where Node Agents will collaboratively and autonomously operate to manage the network based on the defined network goals and operational targets.

2. Literature Review

The idea of software agents monitoring and managing computer networks is not novel. This chapter reviews how other researchers have used software agents to monitor and manage computer networks. This chapter also reviews the advantages and/or benefits of using intelligent agents for network management.

2.1. Introduction

This chapter introduces software agents, network management and software agents in network management. It discusses the languages, platforms used to implement agent systems and introduces the types of artificial rule engines. Further, the chapter reviews traditional methods of network management and discusses their advantages/disadvantages as compared to other technologies. Existing network management systems based on software agents are extensively discussed in this chapter.

2.2. Software Agents

There are numerous definitions for the word ‘Agent’. In telecommunications, an agent refers to any program that acts on behalf of a network administrator and is capable of migrating autonomously from node to node in a network to perform some computation on behalf of an administrator [13]. The basic necessity for this autonomy derives from the fact that an agent must be able to carry out functions in a flexible and intelligent manner that is quick to react on changes in the environment without requiring constant human supervision or involvement. Ideally when multiple agents reside within an environment, they are able to communicate and cooperate to achieve a specific goal [14].

Software agents have the ability to migrate from node to node, learn about their environment, communicate with one another and possess some level of intelligence about their environment of execution. Therefore, they suit an environment of computer devices connected over a network. This dissertation presents a research on multi-agent technology as a substituting paradigm over an existing client/server technique used predominantly for current network management.

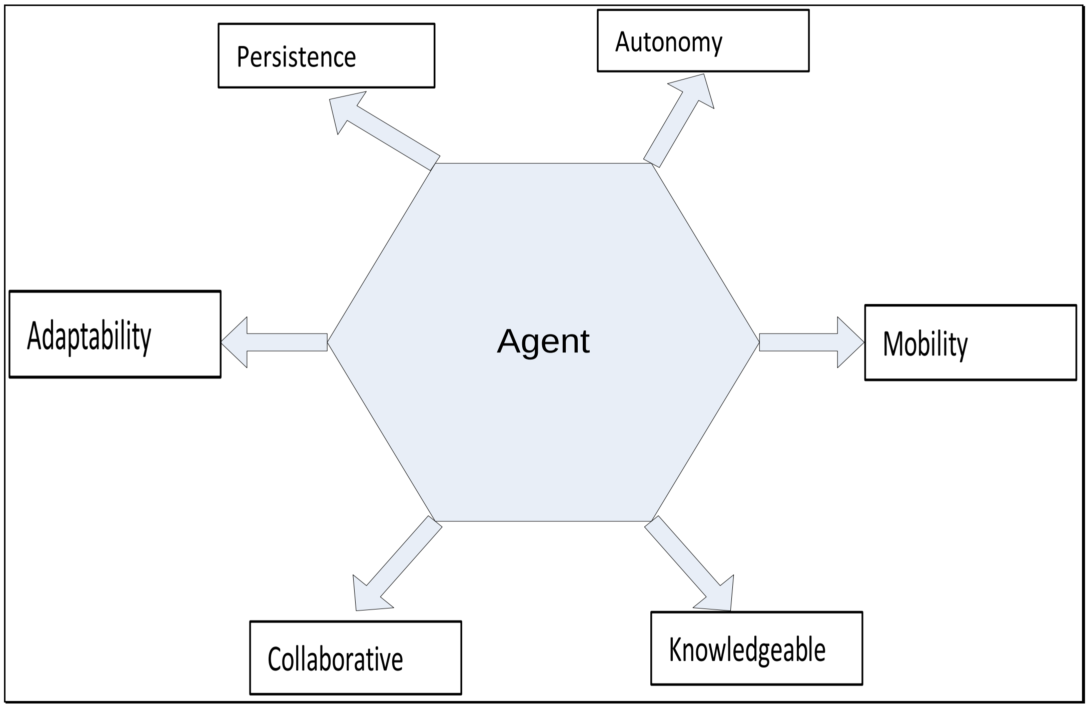


Figure 3: Agent Characteristics (adapted from [5]).

Figure 3 describes the agent characteristics. The characteristics are described as follows [5]:

1. Knowledgeable - Agents are capable of interpreting their goals and knowledge.
2. Adaptability – An agent's behavior may be altered after it has been deployed.
3. Autonomy – An agent is responsible for its own thread of control and can pursue its own goal largely independent of messages sent from other agents.
4. Mobility – Agents have the ability to move from one executing context to another, either by moving the agent's code and starting the agent fresh, or by serializing code and state,

allowing the agent to continue execution in a new context, retaining its state to continue its work.

5. Persistence – Refers to the level to which the infrastructure allows agents to keep information and start over a comprehensive time, counting robustness in the face of likely run-time failures.

6. Collaboration – Agents are capable of communicating and work cooperatively with other agents to form multi-agent systems working together on some task.

Figure 4 shows the Agency Position Representation. The phrase Agency specifies the abstract and physical position in which agents reside and execute [5].

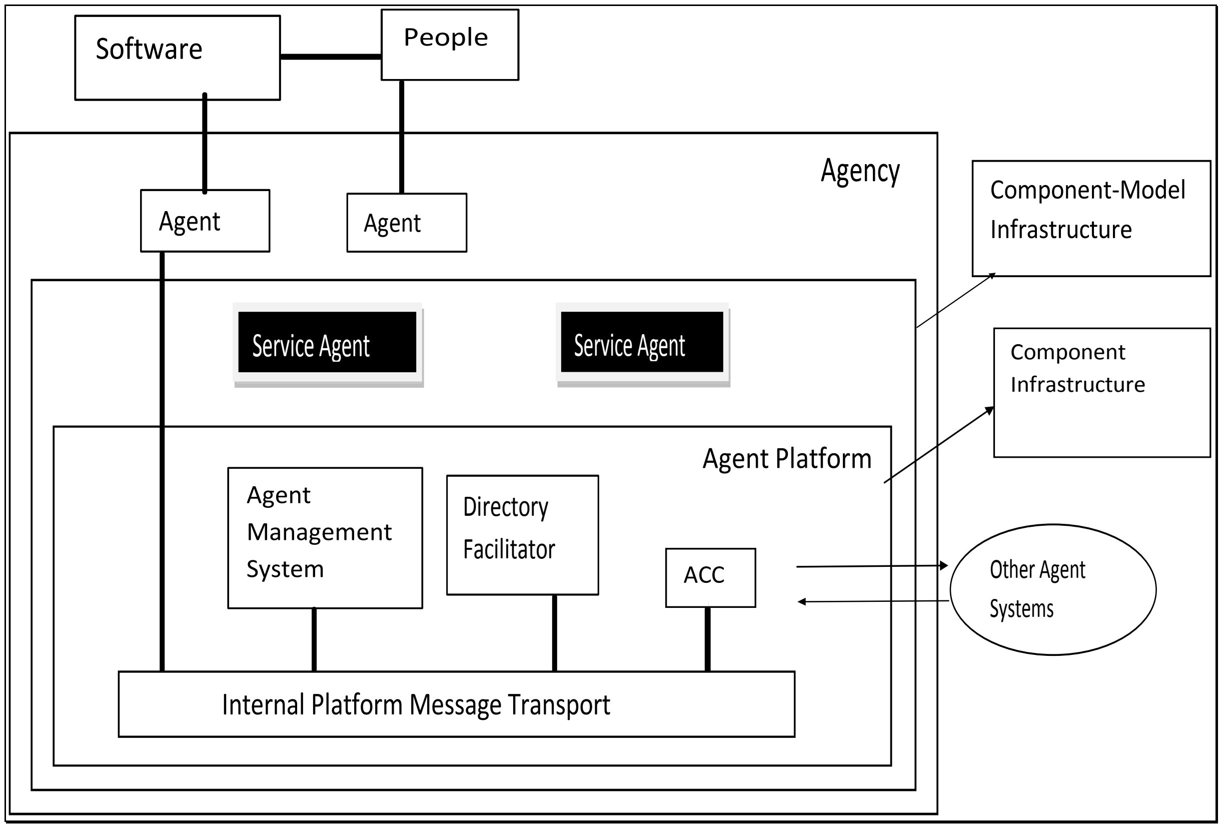


Figure 4: Agent System Architecture (adapted from [5]).

An agent platform is a model structure that offers confined services for agents and means for them to access remote services [5]. An agent system can offer intrinsic services by making use of Service Agents which form the Component Infrastructure. These services include communication, security, naming, persistence, agent management and agent mobility in the case of mobile agents.

Within the Agent System Architecture [5]:

1. Agent Communication Channel (ACC)-routes messages between local and remote Foundation for Intelligent Physical Agents (FIPA) agents, realizing messages using an agent communication language.
2. Internal Platform Message Protocol (IPMT)-provides communication infrastructure.
3. Directory Facilitator (DF) – provides “yellow pages” services for FIPA agents that register agent’s capabilities so that an appropriate task-specific agent to handle the task can be found.
4. Agent Management System (AMS) – controls creation, deletion, suspension, resumption, authentication, persistence and migration of agents. Provides “white pages” to name and locate agents.

2.3. Network Management Systems

This section discusses the types of network management systems, namely: static centralized, static decentralized and lastly decentralized management system.

Static centralized, Figure 5, monitoring mechanism is a mechanism whereby the monitored nodes communicate directly with one single monitoring station. This monitoring station is in charge of collecting, aggregating and processing raw network data. This model is widely used especially by small networks using SNMP [4]. But this mechanism results in processing and communication bottlenecks thereby limiting the number of elements that can be monitored and the rate at which information can be processed. In addition, SNMP favors a polling approach that limits the ability to track problems in a timely manner while requiring management traffic even if no significant change has occurred [4].

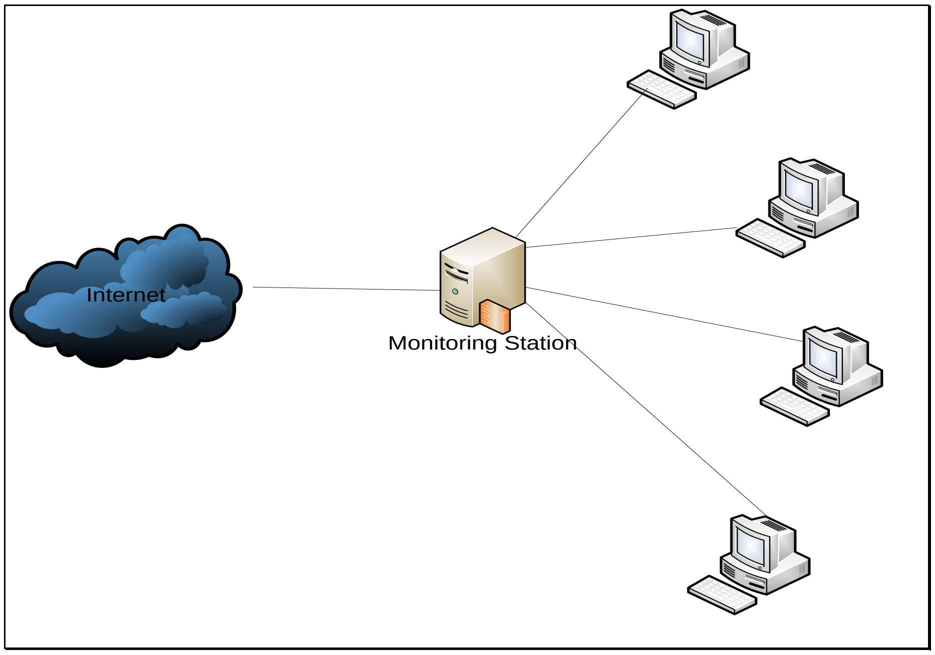


Figure 5: Architecture of a Static Centralized Monitoring Mechanism.

Static decentralized monitoring mechanism adopts a hierarchical management architecture where there are multiple area monitors with one system acting as a main monitoring station. This mechanism can cope with the scalability problem, but still inherits other problems of centralized management and cannot easily cope with frequently changing, dynamic environments. In addition, another type of decentralization found on distributed object technologies such as Common Object Request Broker Architecture (CORBA) and Java Remote Method Invocation (RMI) turned out to be accepted in network management [4]. Figure 6 shows an overview of static decentralized monitoring system.

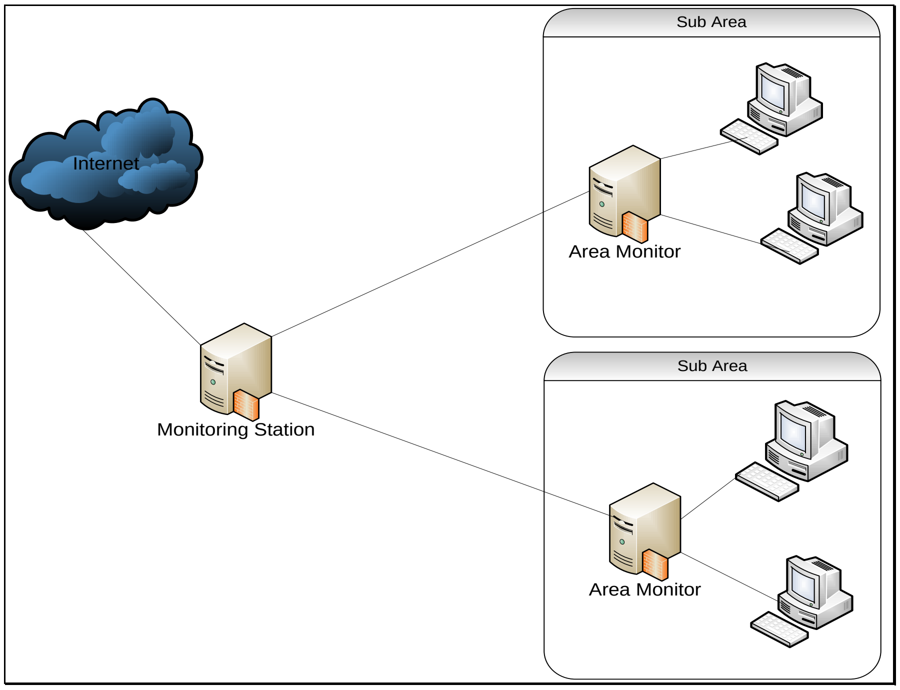


Figure 6: Architecture of a Static Decentralized Monitoring Mechanism.

Thus, in this research project, the use of decentralized and flexible multi-agent based network architecture to monitor and manage rural broadband networks adaptively and efficiently is proposed. A decentralized monitoring mechanism is whereby monitoring functions are dynamically introduced at the node level when and where they are required. This research proposes a system that implements a multi-agent system functionality/idea by deploying agents on network devices and performs network management goals based on the network rules and agent goals specified. In the proposed research, each mobile agent will be generally designed to reside on agent-executable nodes in a network, sense the state of a network device, process the received management information and therefore execute the predefined goals to the network device.

The decentralized monitoring mechanism has been widely used for large scale, high-speed and distributed networks [15]. The use of agents to monitor and manage these systems has proved to be reliable over traditional management mechanisms.

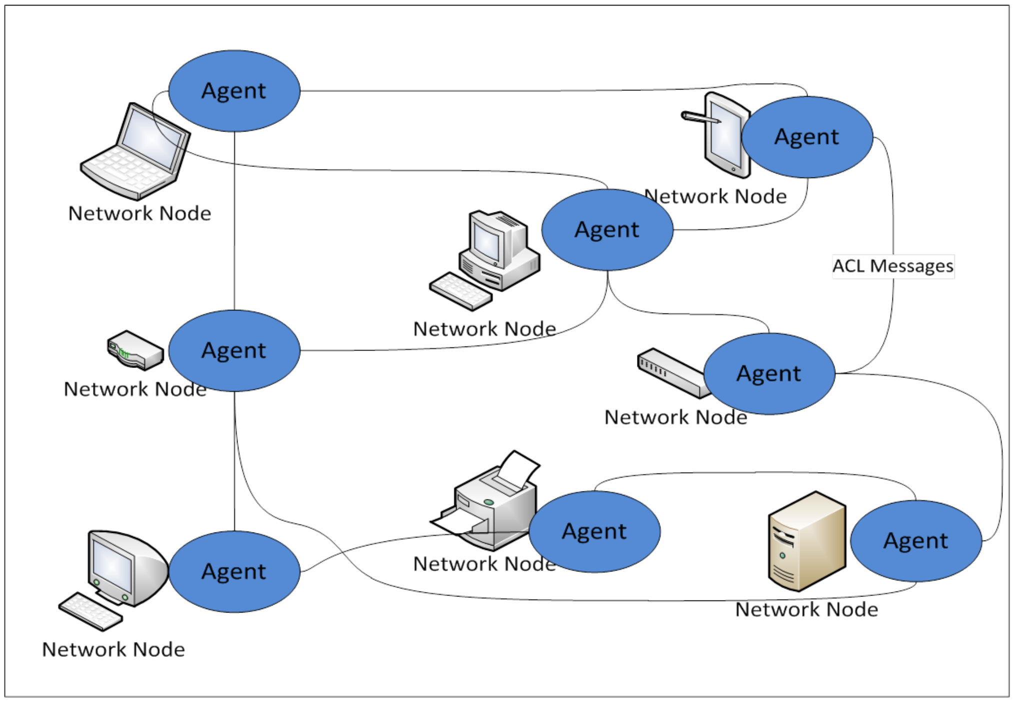


Figure 7: General Architecture of a Decentralized Monitoring Mechanism.

The key advantage of a decentralized management approach, Figure 7, is that it runs as a distributed process instead of a centralized process. As such, it has the potential to satisfy most of the above-mentioned challenges facing rural networks. This system will be fully based on multi-agents, as they have proved to be efficient and effective due to their ability to be autonomous and goal driven [16].

2.3.1. Comparison of Static Network Management Systems

To date, there are a large number of traditional network management systems. This section will compare some of the common systems that exist and discuss their core features and capabilities. This section will conclude by discussing the major features that the proposed system desires to include as its features.

2.3.1.1. OpenNMS

OpenNMS is one of the oldest open source network management systems. It is a Linux package built on Java, Tomcat, PostgreSQL and RRD Tool [17]. OpenNMS is a platform independent NMS that has the ability to manage thousands of devices and show the network statistics in a web-interface. This system provides support for IPv6 throughout, automatic network/node discovery, and event management and notification features [17]. OpenNMS offers integration features such as integration with HypericHQ NMS and JBoss Drools Expert for event correlation and SNMP protocol and JMX technology [18].

2.3.1.2. OpManager

This is a commercial NMS that is managed via a Web Graphical User Interface (GUI) that runs on Windows machine. OpManager is a full NMS that offers advanced combination of Wide Are Network (WAN), Server, Application monitoring with integrated help desk, asset management and WAN traffic analysis functionality [19]. Further, OpManager offers an easy-to-use interface that lets a network administrator to specify network policies across multiple devices efficiently. This NMS presents an advanced performance management for critical network resources such as WAN links, firewalls, routers, switches, VoIP call paths and other network infrastructure devices [20]. Figure 8 shows a static decentralized approach of monitoring whereby the probes gather information and send it to the central server for processing and decision-making [20]. These probes are dedicated managers located in sub-networks.

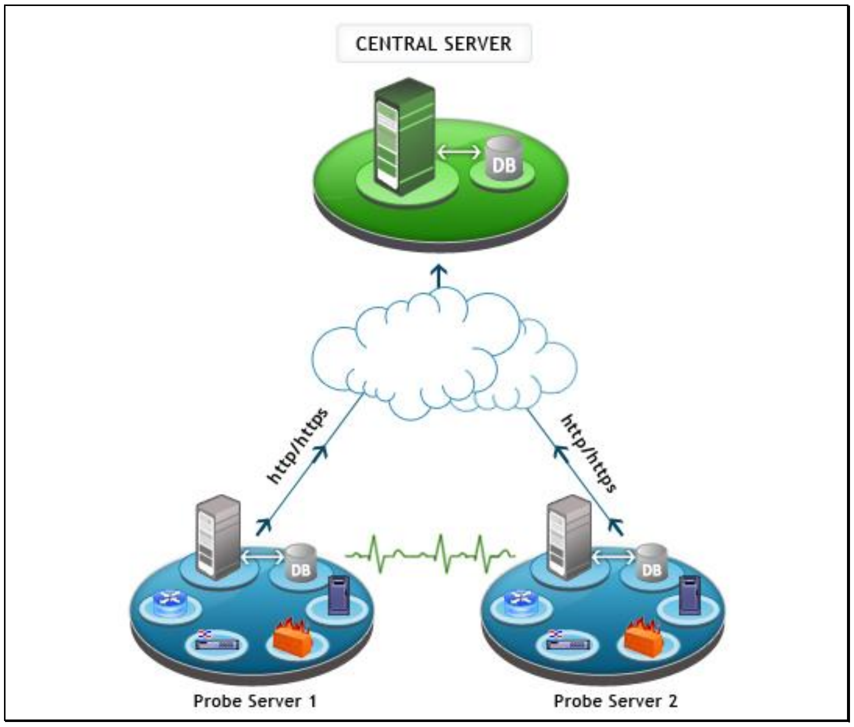


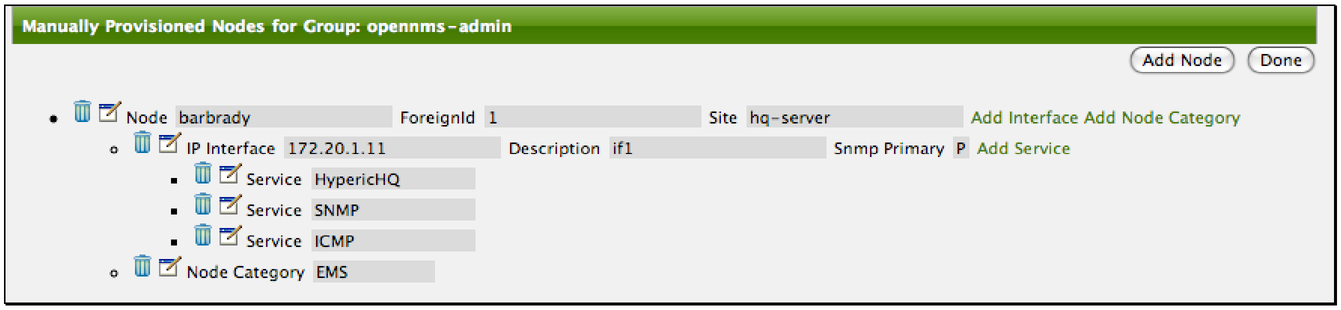
Figure 8: Probe-Central Architecture of OpManager [20]**.**

2.3.1.3. Nagios

Nagios is one of the most widely implemented open source NMS and allows to gather network performance and availability information from any platform. Even though its GUI is carelessly designed, the installation of Nagios is straightforward and it complements Linux proven standards. One identified drawback of Nagios is that, it requires someone who is familiar with Linux operating system because when adding a new device in the network it needs one to manually edit configuration files from Linux operating system [21]. This makes it difficult for real world Internet Technology (IT) organization to use it for their network management task. Nagios agents help to spot problems before they occur; they immediately know when problem occurs and easily detect security breaches [22].

2.3.1.4. Hyperic HQ

Hyperic HQ is a NMS completely written in Java and is deployed on JBoss Application Server. This NMS is mostly used for discovery purposes, such as device vendors, Central Processing Unit (CPU) states, hard disk memory and running application as well as network states [23]. Hyperic HQ can easily be integrated with OpenNMS; it is used as discovery application that views alerts and notifications through a configurable web portal. This NMS comes in two editions, Enterprise Edition and Open Source Edition. The enterprise edition is developed for large-scale companies who run critical web applications and systems. And the open source edition is developed to provide all basic management facilities for web applications and IT infrastructures.


Manually Provisioned Nodes for Group: opennms-admin

- Node barbrady ForeignId 1 Site hq-server
  - IP Interface 172.20.1.11 Description if1 Snmp Primary P
    - Service HypericHQ
    - Service SNMP
    - Service ICMP
  - Node Category EMS

Figure 9: Provisioning HQ Server in OpenNMS [18]**.**

Figure 9 shows a sample of integration of Hyperic HQ in OpenNMS and it also supports the monitoring service of HQ itself and its called HypericHQ [18].

2.3.1.5. GroundWork Monitor

GroundWork Monitor is a NMS used to monitor enterprise business applications on premises or in the cloud [24]. This NMS can easily be integrated with some management tools, such as Cacti and Nagios, and Network Mapper (NMAP). GroundWork Monitor has two editions, the enterprise edition and the open source edition. Both these editions have Web GUI that does not conform to marketing claims in terms of usability.

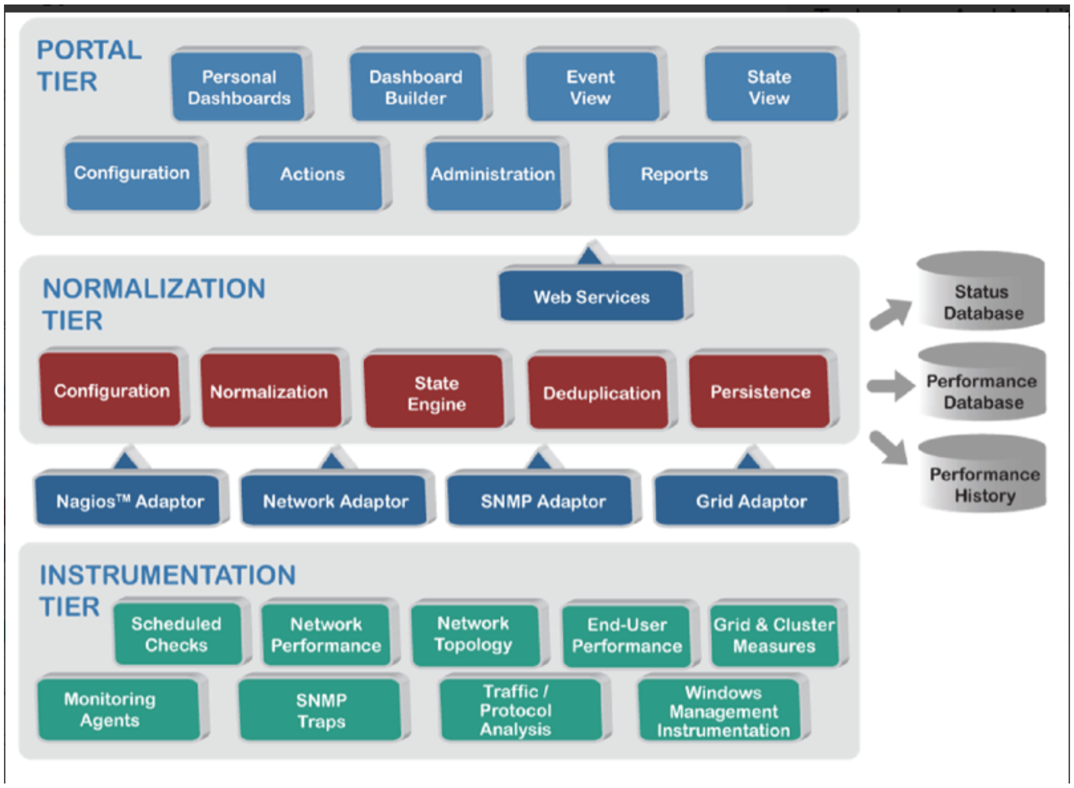


Figure 10: Three-Tiered Architecture of GroundWork Monitor Enterprise Server [24]**.**

Figure 10 depicts a three-tiered architecture of GroundWork offered in the enterprise edition. The Instrumental tier is responsible for data gathering that is basically done by Cacti and Nagios. The Normalization tier is in charge of storing the data gathered in a normalized structure and present it to the Portal tier through web services. The tier that is used for visualization of graphs, network performance data, network status and real-time display of events is the Portal tier [24].

2.3.1.6. Argus

Argus monitoring system, Figure 11, is entirely developed in Perl and is a platform independent NMS [25]. Argus is developed to monitor network status and hardware devices on a network through a web-based interface. Argus is an open source NMS and its web interface is easy to use, it provides a basic alerting interface whereby, red color points to an error and yellow color refers to good functionality of a network. Argus provides support for both IPv4 and IPv6 and can manage thousands of network devices. This NMS handles MySQL requests easily and presents the results in a graphical user interface [25].

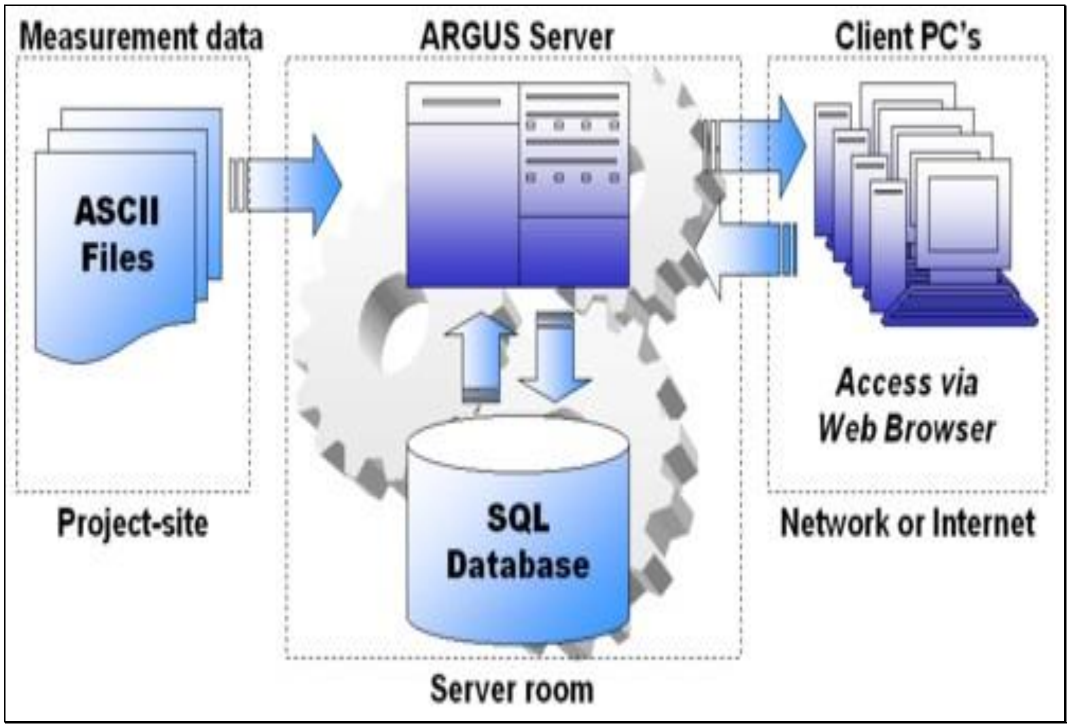


Figure 11: Argus Design [25]**.**

2.3.1.7. Cacti

Cacti is a cross platform NMS that is written PHP and PL/SQL and uses RRDtool for the network graphing solutions [26]. Cacti software provides a usable web interface that graphs network resource utilization, CPU states and network traffic. Cacti software supports the ability to retrieve network data using SNMP through PHP scripts that are used to update RRD files. This software allows an executive administrator to create different levels of user permissions for other administrators through its interface [26].

2.3.1.8. SNMPc

SNMPc is the first Windows based NMS that has a support for IPv6 and secure SNMPv3. This NMS has device limit of 25000 and provides both local and remote access using a remote console application from any Windows machine using a local and remote TCP/IP connection [27]. SNMPc is a commercial, secure distributed NMS that allows real-time network monitoring. SNMPc software supports the automatic layout of a network map in a hierarchal form and each map object (network device) can be selected to view the object current state. This graphical network map is called Map Navigation Tool Window and allows users to zoom in/out to view a set of devices [27].

2.3.1.9. NetXMS

NetXMS is an open source network management and monitoring system with the core server running on Windows or Linux. Compared with other NMS servers, NetXMS has the largest database server that embeds MS SQL, IngreSQL, MySQL, Oracle and SQLite. NetXMS offers complete network management and monitoring with graphing of network infrastructure [28]. NetXMS has a separate web interface that helps the network administrator to easily add new devices and configure network changes and a different web interface for basic web browsing. This NMS has dedicated agents for node discovery, alerting and reporting network errors. With the support of large database, it would make it easy for a network administrator to grow a network if need be [28].

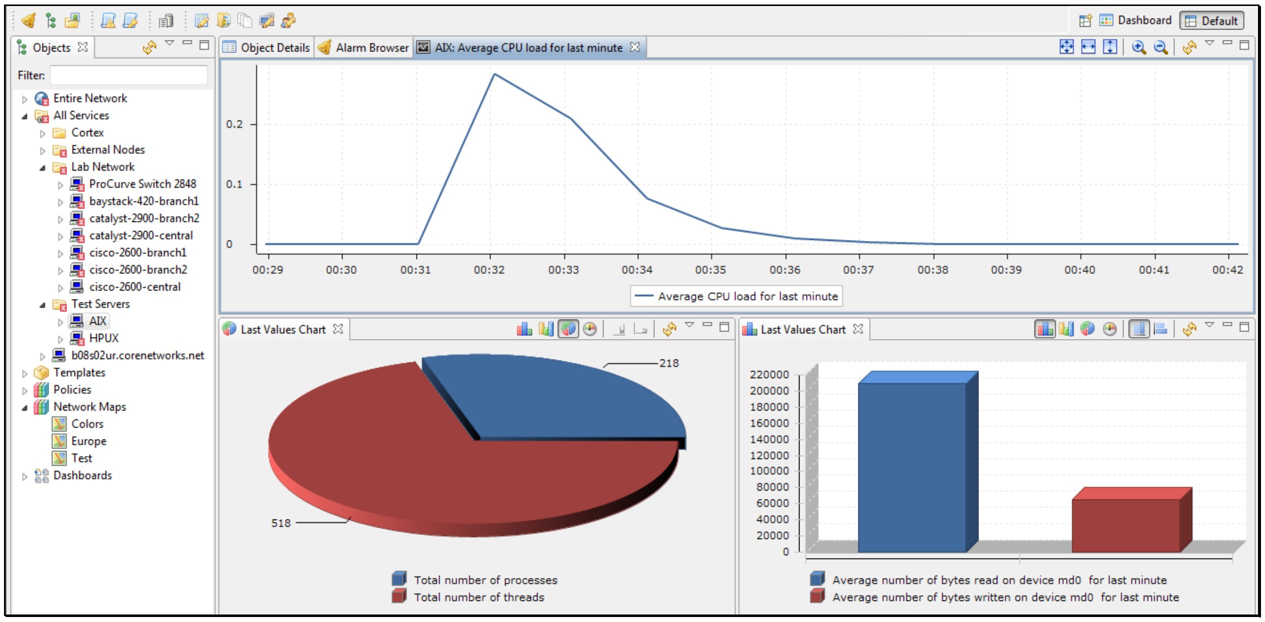


Figure 12: NetXMS Management Console [28]**.**

Figure 12 shows a graphical user interface of NetXMS with graph representation of average CPU times and network statistics.

2.3.2. NMS Comparison

Table 1 depicts the comparisons of the traditional NMSs based on the following features and capabilities:

- Auto-discovery: ability for the system to automatically determine added network devices
- Trends: ability for the system to show network statistics over time
- Distributed monitoring: system that offers multiple servers to distribute the load of network monitoring

- Inventory: ability for the system to keep records about network device hardware and software information and network user’s information

- Platform: a necessary requirement for the system to be installed on

- Data storage method: method used to store network and user information it monitors

- Triggers or alerts: ability for the system to detect when thresholds are reached thereby alerting network operator

- Agentless: the reliability of the system on agents to monitor network nodes and sending back and forth to the central server

- Maps: graphical representation of the network devices and the links between them

- Access control: an administrator should be able to define access to certain parts of the system as per-user or per-role basis

- Web application: a system that offers network statistics in web-based front end, allows the viewing of notifications and also allowing a full control of notification maintenance through the web-based front end

Table 1: NMS Comparison.

Feature	Open NMS	Op Manager	Nagios	Hyperic HQ	Argus	Cacti	NetXMS	SNMPc
Auto Discovery	X	X		X			X	X
Trends	X	X	X	X	X	X	X	
Distributed Monitoring	X	X	X		X	X	X	X
Inventory	X			X		X		
Platform	Java	Java	C, PHP	Java	Perl	PHP	C++	Java
Data Storage Method	JRobin, Postgre SQL	MS SQL, MySQL	Flat File, SQL	Oracle, MySQL, Postgre SQL	Flat File, Berkeley DB	MySQL, RRDTool	MySQL, MS SQL, Oracle	MIB

Triggers/ Alerts	X	X	X	X	X	X	X	X
Agentless				X	X	X		X
Maps	X	X	X	X			X	X
Access Control	X		X	X	X	X	X	X
Web App	X	X	X	X	X	X	X	X

Trending is the ability of the NMS to provide network data over a period of time. Distributed monitoring refers to the ability of an NMS to distribute the load of network monitoring to one or more servers. Agentless states that the NMS does not have an agent that resides on managed devices to send the monitoring data to the central server. From the comparison in Table 1 all the NMSs support web based interface and a system that send alerts to the administrator. According to Table 1, few NMSs support the inventory of storing hardware and software information of managed devices and some support the auto-discovery of network devices.

From the analysis of the features and capabilities of the NMSs profiled above, the following common NMS functions were extracted:

- Data storage and analysis of network statistics;

- Inventory of managed network devices to store their hardware and software information;
- Web or graphical user interface to view network statistics;
- Event management and alert notification tool; and
- NMS that can operate in any network device regardless of the underlying platform.

2.4. Multi-Agent based Network Management Systems

The technology of MAS in managing networks has in recent years drawn many researchers' attention. There is a significant number of noteworthy research carried out on software agents for network management [29]. Most of these projects have focused on large-scale networks, distributed networks, complex modern technology systems such as WANs and heterogeneous networks. For example, ExperNet is a multi-agent based network management system that was developed to manage a WAN in Ukraine [30]. ExperNet assisted network administrators to quickly identify errors and suggest solutions through a Web GUI. The implementation of this system was based on SNMPv2, which made it easily applicable to any network. This system is made of software agents that are capable of both local problem solving and social communication among them for synchronizing problem analysis and repair. For the implementation of the agents, distributed Prolog Ⅱ system enhanced with networking capabilities was used. ExperNet has been developed, installed and tested successfully in an experimental network in Ukraine [30]. The hierarchical architecture of ExperNet shows that the system was developed with many networking modules to provide robustness and adequate performance. It used DEVICE which is a knowledge base system, Big Brother a host-monitoring tool, HNMS+ a network-monitoring tool with many user interfaces between them [30].

In 2000 Marcus assessed the significance of mobile agents by developing simple prototype applications, from design to implementation and testing, using both a 'Traditional' client/server based approach and by utilizing a mobile agent architecture [31]. He used Java to develop a mobile agent oriented application that is similar to the prototype application.

In 2003 Maj et al. developed a system that dynamically reconfigured a network using software agents [32]. Network Reconfiguration includes routing changes as well as constructing new and removing inactive links between network nodes. In 1998 Kim et al. investigated the use of Artificial Intelligence techniques to manage large-scale high-speed networks [16]. They developed a network management system that efficiently and effectively monitored and controlled network resources in a large network. The system was called ExNet and provided a web interface to view network statistics and expert systems' recommended actions. Kim et al implemented a rule-based prototype of ExNet and the incorporated ExNet modules with IBM NetView network management system. NetView is the program offers real time monitoring and active testing on servers supported by SNMP [33]. [34] Kim *et al* proposed a network monitoring mechanism that is based on flow management to efficiently manage a large network in real time. They argued that the mechanism can efficiently detect Denial of Service (DoS) attacks, port scans and worm propagation based on simulation with the network traffic. In 2003, Lefebvre et al presented a framework based on mobile agents that managed heterogeneous networks [35]. Even though their project focused on the framework, they also presented an example of mobile agents that are able to locate a fixed set of network failures and detect the possible cause of the failure accurately. Their experimental results showed that mobile agents could easily execute some network management tasks.

In 2010, Mitrovic et al investigated a solution to improve multi-agent systems to be fault tolerant by introducing types of mobile agents, *ConnectionAgent* and *RemnantAgent. RemnantAgent* was built to track the path of agents and *ConnectionAgent* was developed to build and maintain reliable networks of dispersed multi-agent systems with both these agents being capable of an easy integration with any multi-agent system [36]. Every network administrator would want to experience a management system that is self-healing and easily monitor a newly configured device according to organizational policies and standards. In 2004, Tripathi et al. presented a multi-agent based NMS that simply detects any malfunction of a network and provides mechanisms for self-recovery should there be failing network components [37]. This multi-agent based NMS was developed for large-scale network and was named as the Konark monitoring system. The Konark NMS was implemented using Ajanta framework that provides three main components; Agent, Agent Server and Agent Registry. One of the most important features of network management system is to be able to manage available network resource effectively. Marzo et al have presented a distributed design for managing network resources effectively. The architecture of bandwidth management task showed how the bandwidth re-allocation and re-routing of logical paths from logical work path to a backup path happened [38]. This system used fixed and distributed software agents over the managed network devices; they argued that their system was meant for backbone and core networks and did not find a reason to use mobile agents. They also argued that backbone and core networks have high bandwidth and are more reliable hence it was not valuable to add the complexity of mobile agents [38].

The most significant issue about these dynamic management systems is that, they all focus on large-scale networks with minor differences in the language used to develop a system or rule engines used for knowledge processing and decision making and types of the Agent framework used. Their intention is courageous and has proven to be reliable, but the service these systems provide to large-scale networks and the challenges faced by the rural networks, leads to the view that it makes perfect sense that the dynamic management and monitoring mechanism be used in SLL. The next section discusses the agent development approaches based on mobile code.

2.5. Approaches Based on Mobile Code

There are three types of Mobile Code for network management namely: Code on Demand, Remote Evaluation and Mobile Agents. These approaches have existed for a long time and provide advantages over the client/server architectures, thereby eliminating limitations that come the with client/server approach [39]. The advantages and limitations of each of these approaches are as follows:

2.5.1. Use of Code on Demand

This approach implies that agents be developed in a manner that they only have minimal/necessary functions. It is based on a theory that, not every network node requires the same management functions. Therefore, when a node requires some special functions that are not provided by an agent code, this function will be downloaded from a central server/code server and be dynamically introduced to an appropriate node. This approach claims that it is not necessary to statically include all the management functions on an agent because it leads to a waste of resources[39].

2.5.2. Use of Remote Evaluation

Remote Evaluation is different from the Code on Demand approach in the sense that it does not use agents as managing entities although the NMS has a dedicated piece of codes to perform management functions when needed. When a network node has a request, the NMS sends a piece of code/develops it, and then sends it to the node to perform the requested task. This issue addresses the problem of bandwidth wastage that comes with the central NMS, because it only sends a piece of code that has the management function when needed. In a centralized NMS, the management functions are done on site by transferring variables involved to a specified node. In this system, the NMS uses a polling approach to constantly check on a network performance on a specified time interval. Therefore, the higher the number of requests of variables involved, the larger the amount of bandwidth occupied on the network. This approach proposes that the network management function should be operated directly on the device, only when needed [39].

2.5.3. Use of Mobile Agents

In the above-mentioned approaches, the central server is always involved one way or another where the execution of management functions is required. With mobile agents, the network management function is different and adds a level of autonomous and intelligence approach to managing network resources. In this approach, the network monitoring and management tasks and network policies are explicitly specified to an agent. This allows network management to be a delegated process because these agents have knowledge of what to do when an error/failure occurs. Mobile agents support asynchronous communication and this is beneficial to rural networks where there are unreliable links between the managed nodes and the central server. The following section describes the benefits of mobile agents in network monitoring and management [39].

2.6. Benefits of Multi-Agents in Network Management

This section discusses the main benefits of software agents for network management as compared to client/server model. Client/server model has been around for a very long time and there are network infrastructures where this model suits the management function. However, when the network infrastructure faces challenges such as experience on rural networks (e.g. having intermittent energy supply and power problems, distributed over a geographical area, and growing and expanding increasingly) this model needs to be reconfigured manually. When one failure occurs on the central device, all the managed devices experience a disruption. The MAS paradigm supports the decentralized management functions at a node level to ensure high QoS and self-recovery. MASs have their own requirements for execution and these include:

- Agent Execution Support - platform that allows agents to operate and provide basic agent services;
- Management Support - means to manage agents in the agent platform for example start/stop agents, pause/resume agents tasks, kill/create new agents;
- Security Support - agent platform must secure agent communications and system agents from external intruders;
- Mobility Support - if agents are to migrate around their environment, the agent platform

must provide methods for their movement;

- Unique Identification (UID) of Agents - in an agent society there is a need to uniquely identify agents of the same type; and

- Communication Support - if agents are to collaborate or compete, they have to know how to communicate so they achieve their main goal.

MASs have the ability to reduce latency because they lower delay times by processing node requests efficiently. MASs are capable of having asynchronous communication and operation with their master agent/server computer. Software agents need not communicate with the server side computer after deployment. Therefore, even if the server is shut down or the network between them is disconnected, they will still carry on with their computation. One of the aims of utilizing MASs is to reduce bandwidth utilization. This can be achieved by installing agents directly on network devices, rather than constantly sending requests back and forth over the network. This is based on the theory that these requests would use more bandwidth than placing an agent on a node plus the communication amongst them. To provide maximum throughput and minimize response times, MASs provide a mechanism for load balancing. This is achieved by the idea that agents are able to track their path of execution and their ability not only to learn about their environment but can repeat the steps that would avoid overload [40]. Dynamic deployment of software agents is useful so that they can take their own decision when they arrive at a destination of execution [41].

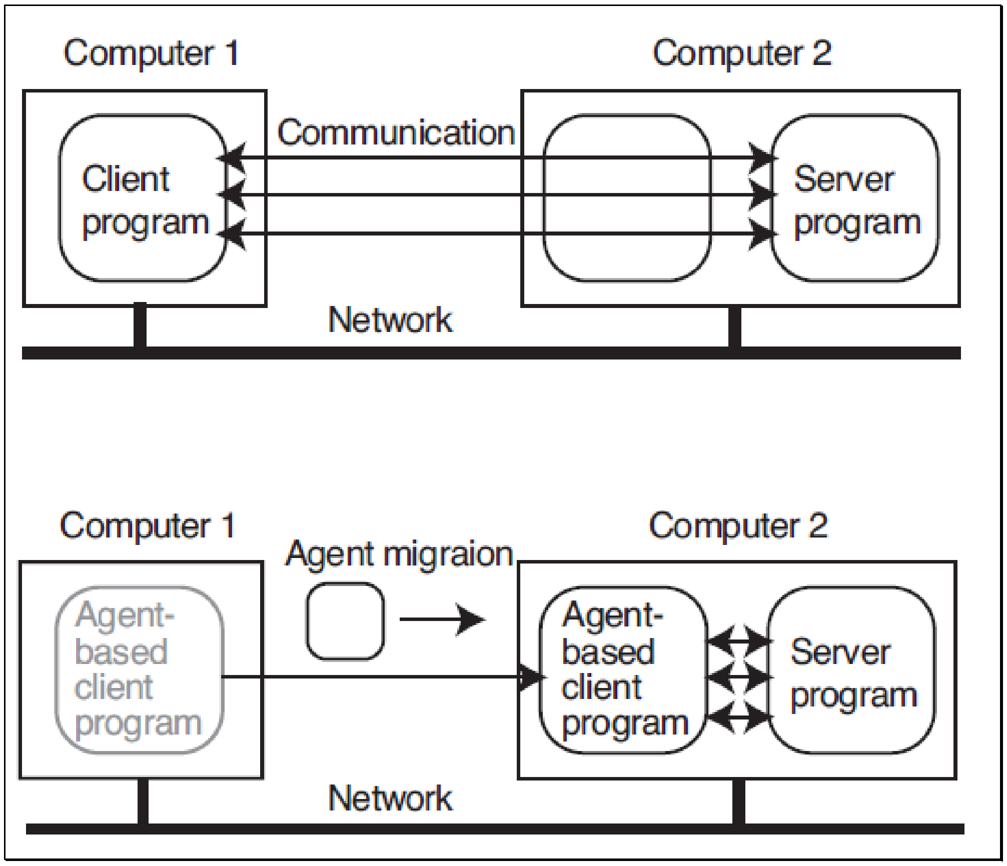


Figure 13: Reduced Agent communication [41]**.**

Figure 13 shows the difference between the client/server agent-based architecture and how agents reside on the network thereby having minimal communication with the agent platform.

2.7. Application Domain of Multi-Agents

Multi-agent systems have been of great interest since the early 1980s and increasingly cover a wide range of domains [42]. MASs are mostly used for data collection, searching, filtering, monitoring and negotiating, entertainment and information dissemination, among other applications differing by domain services.

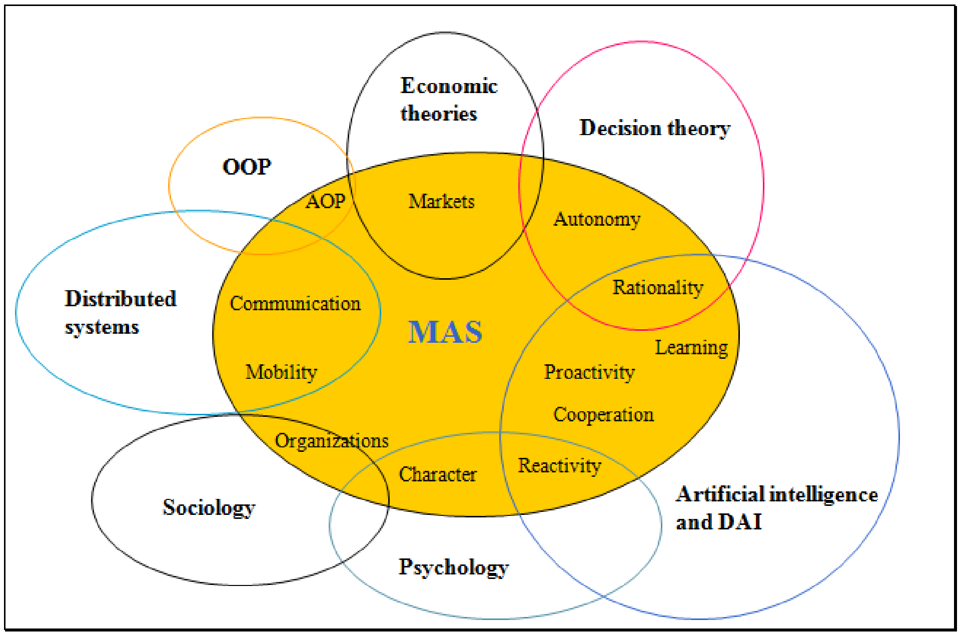


Figure 14: MASs with other disciplines [43]**.**

MASs encompasses a wide range of fields. From Figure 14, agents assist in e-Learning in which they act as a peer-learning tool in giving advice and participate in simulation. MASs can also act as a negotiating tool in trade markets through the facilitation of the sell-and-buy method, taking decisions based on a wide range of variables, and making and cancelling orders. These systems are used in real world systems such as electricity distribution systems, transportation systems, e-commerce, Human-Computer Interaction (HCI) and game theory [44].

2.8. Artificial Intelligence Rule Engines

A Rule Engine or Inference Engine is a program that has a capability to make expert systems to reason about the information in the knowledge-base for decision making [45]. A rule engine has three main components[45]:

- Ontology-presentation framework of concepts in the world and their relationships;
- Rules-to perform the reasoning and facilitate the decision making process; and
- Data-working memory that consists of facts about the environment of execution.

The basic function of a rule engine is to match facts and data with the rules provided by a system developer to perform conclusions consequently taking actions. For example:

```
if
    <user has exceeded 8 hours of work per day>;
then
    <shut down users machine>;
```

The rule engine is responsible for matching existing or new facts against the rules and this process is called pattern matching. Figure 15 shows a high level view of processes that occur in a system that uses the rule engine.

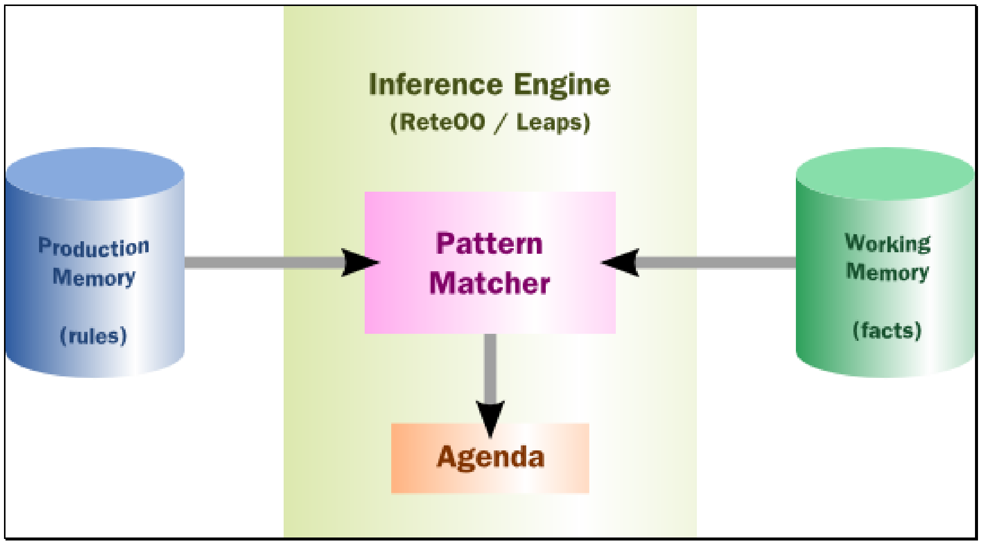


Figure 15: High-Level Architecture of an Inference Engine.

The rules are stored in the production memory and the facts are stored in the working memory, and these facts can change. When there is a large number of rules and facts, there is a need for Agenda. An Agenda is used to solve any conflicting rules using a Conflict Resolution strategy. In the next sections I discuss some of rule engines used to develop reasoning expert systems.

2.8.1. OpenRules

OpenRules is an open source rule-based development AI engine that has Java & .Net integration. It supports collaborative rules management. OpenRules offer the following functions [46]:

- Rule Repository – for management of enterprise-level decision rules;
- Rule Engine – for rule execution;
- Rule Learner – for rule discovery and predictive analytics;
- Rule Solver – for solving constant satisfaction and optimization problems;
- Finite State Machine (FSM) – for event processing and collecting the dots; and
- Rules Dialog – for building rule-based web questionnaire.

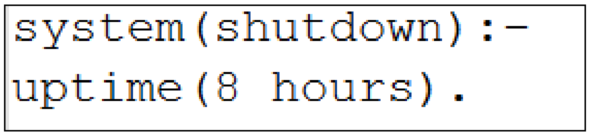
OpenRules consist of decisions, decision table (i.e. if-then condition); glossary for decision variables; data that specifies concrete test instances; method specification that is on Microsoft (MS) Excel or Java based and environmental variables (i.e. imports).

2.8.2. Jess

Java Expert System Shell (Jess) is a rule engine that is both an open source for academics and the license is also available as commercial software for enterprise companies or individual purposes [47]. Jess uses a forward-chain method with enhanced version of the Rete algorithm to process the rules. Rete algorithm is a mechanism for solving difficult many-to-many matching problems [47]. Jess is a scripting language that allows users to develop expert systems that have the capability to reason using the knowledge supplied in the declarative form and gives developers full access to all Java Application Programming Interfaces (APIs). Jess has its own declarative language called Jess Markup Language (JessML). JessML allows users to define the rules as well as defining and calling methods. This support of an Extensible Markup Language (XML)-based rule language makes it easier to transform other XML languages to Jess and vice versa.

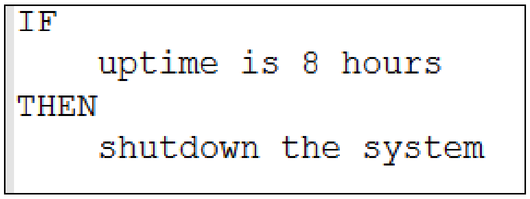
2.8.3. Prolog's Inference Engine

Prolog has a built-in backward chaining rule engine that can be used to partially implement some expert systems [48]. Prolog inference is used to derive conclusions from the knowledge provided as rules in prolog language. Prolog's Inference Engine format [48]:



```
system(shutdown):-
uptime(8 hours).
```

Using the normal IF THEN format, this rule is [49]:



```
IF
    uptime is 8 hours
THEN
    shutdown the system
```

Prolog's rule engine starts from the conclusions that will be drawn to the rule, meaning it uses a backward-chain method. The advantage of a backward-chain method is that, it easily solves the structured selection type problems. The shutdown system example is based on information that might be absolutely false or true, therefore, the rule engine has to be certain and only trigger the shutdown command of the system only when the user has exceeded 8 hours [49].

2.8.4. Drools JBoss Rules

Drools JBoss Rule is a declarative, rule-based engine that is based on Drools; it uses a forward chaining method enhanced with Rete algorithm. JBoss Rule allows developers to focus on stating only the goals of an expert system and not on how to achieve them. JBoss Rule is a declarative rule engine that is easily written in Java, MVFLEX Expression Language (MVEL), Python and Groovy. JBoss Rule is a declarative rule engine that is easily written in Java, MVEL, Python and Groovy, for:

```
rule "User must not use over 250MB in 60 hours" when
    a: Application(megabytes>250)
then
    denyUser(a) {allowInternet = false}
end
```

2.9. Elements of an Agent Platform

The main element of a mobile agent platform is an agent execution environment that provides the fundamental platform-level services and must always be active on the system before agents can be executed. The aim of an agent execution environment is to facilitate the instantiation, retrieval and dispatch of agents [50]. It also acts as an interface between incoming multi-agents and the fundamental system resources and offers a set of services required by the multi-agents to perform their distributed functions. Multi-agents consist of three basic parts: the code that defines the functionality of a multi-agent system; data that is the constant state of an agent and the thread of execution. Agents are generally designed to reside on devices or move within their environment of execution so they can directly invoke changes on the network devices they are installed on. The agent platform provides two forms of migration for each agent, the *strong mobility* and *weak mobility* [51]. Strong mobility enables the transfer of all three parts on every agent migration and weak mobility is whereby there is a transfer of only the code plus the state of information [51].

2.10.1 Elements of JADE Agent Platform

This section briefly discusses the elements of Java Agent DEvelopment framework (JADE) and Figure 16 shows the general JADE agent platform architecture. The JADE agent platform offers fault tolerant service, agent directory and location service, security service, AMS and communication service. Fault tolerant service ensures that agents are able to move around their environment reliably and their state of information is not lost when there is a system or network failure [50].

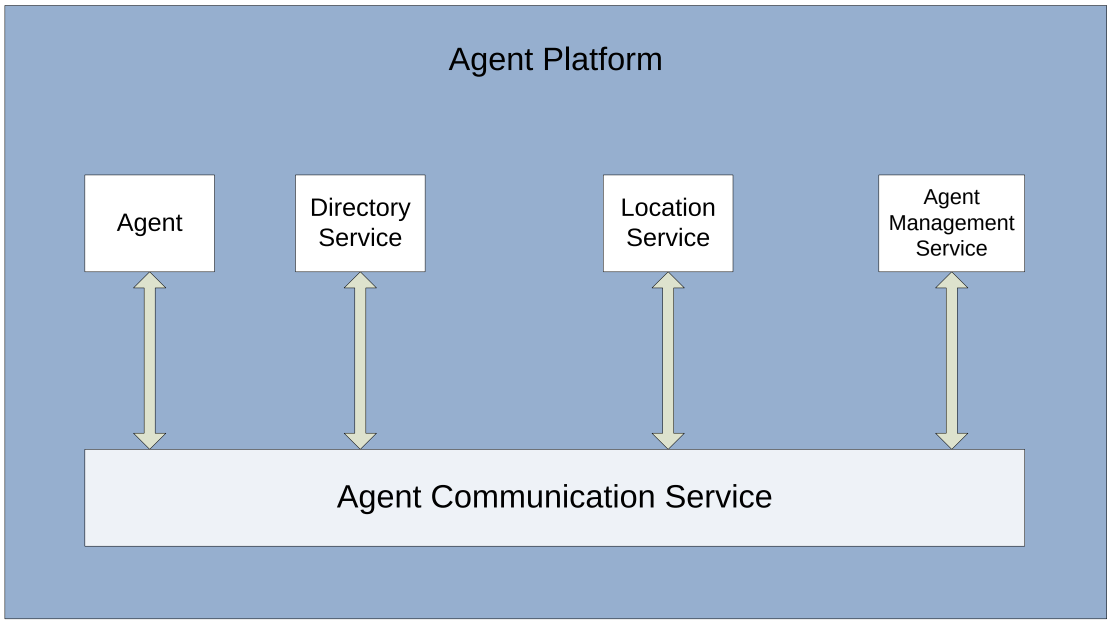


Figure 16: General Agent Platform Architecture [43].

Agent directory allows agents to know the existence of other agents and their services. Agent location service allows agents to track their coordinates and to track the location of information to update or perform certain tasks. In a system that is connected via the network, there is need for providing solutions for security issues such as authentication and authorization. JADE agent platform offers a service to ensure integrity, confidentiality, authentication and access control of the system hosts and users. AMS is an agent (master agent) that has control over all agents' access to and use of the agents in the platform. All agents are registered to the AMS and are assigned unique Agent Identifier (AID). This platform also offers communication service for agents inside the platform and platform agents with external agent platforms. This communication service is facilitated by exchanging messages via the Message Transport Protocol (MTP) and uses FIPA Agent Communication Language (ACL).

2.10. Multi-Agent Programming Languages and Platforms

There are a number of languages that agents use to communicate with one another, namely: Knowledge and Query Manipulation Language (KQML), FIPA-ACL, ARMITIS COmmunication Language (ARCOL), Knowledge Interchange Format (KIF) and Domain independent COOrdination Language (COOL) [52]. KQML is a language and a protocol that enables agent application to communicate together. KQML uses speech-act which defines a set of communication actions such as reply, tell, un-still, sorry and deny [52]. FIPA-ACL is a standard language for agent communication; it also relies on speech-act performative. FIPA standards set how these agents should communicate together [43]. ARCOL is an agent communication language used in ARTIMIS agent technology developed by France Telecom [53]. COOL is an agent communication language that uses speech-act performative. COOL allows system developers to identify agents and manage the communication between agents [54].

The emergence of MAS has led to the development of programming languages and platforms that are suitable for the implementation of these expert systems. For every relevant programming language of MASs, here are the few basic requirements that each language must possess [55]:

- Support of delegation at the level of goals-agents is designed in such a manner that does not define what to do but they are given specific goals and not providing a method on how to reach those goals;

- The language should provide support for goal-directed problem solving. Agents should be able to act to achieve the delegated goals;

- The language should lend itself to the production of systems that are responsive to their environment;

- The language should cleanly integrate goal-directed and responsive behavior; and

- The language should support knowledge-level communication and cooperation.

MAS programming languages are classified into three categories [55]:

- Declarative Style Agent-Oriented Programming Languages is a programming paradigm that describes what requires to be done, rather than describing how to achieve it. Computational Language for Autonomous Intelligent and Mobile Agents (CLAIM), FLUX, MINERVA, DALI and ReSpecT are declarative languages that are reviewed in this section. CLAIM is high-level programming language for mobile agents that uses Himalaya framework [56]. CLAIM allows the development of mobile agents distributed over a network because it supports the migration of agents in an encrypted and persistent state to their destination [57]. FLUX is a programming language for agents that reason logically about their actions on the basis of Fluent Calculus [58]. FLUX has a method of sensing incomplete knowledge base information. MINERVA is a logic programming language that consists of several specialized agents, performing various functions while using and manipulating a common knowledge base [56].

- Imperative Style Agent-Oriented Programming Languages is a programming paradigm that explicitly describes implementation algorithm commands to a program to perform. JADE is imperative [55].

- Hybrid Style Agent-Oriented Programming Languages is a programming paradigm that specifies dependencies in a declaratives manner but includes an imperative list of actions to take as well. Practical Agent Programming Language (3APL) and Abstract Agent Programming Language (2APL) are the two examples of Hybrid Style programming languages [55].

2.11. Conclusion

This chapter reviewed network management by making use of MASs, and in all the related work reviewed, none of them shows the use MASs in rural networks. With all the advantages of MASs for networking monitoring and management, there is little work that has been done in rural areas. There is a decent motive that MASs can address a lot of rural network challenges mentioned in this chapter. In the next Chapter, the system design of the next MAS based network management tool for SLL will be presented.

3. Requirements Analysis and Design

This chapter describes the different types of agent development methodologies and an analysis of the identified research methodologies. It also takes advantage of the literature review in Chapter 2 and specifies the system requirements. Before the chapter concludes, it outlines the system architecture and discusses the system operation.

3.1. Introduction

Developing a Multi-Agent based system requires many technologies to be combined effectively to produce one working system. This chapter discusses how these technologies are used to design the system. It also specifies the requirements of the system (user, system and project requirements) and reveals the low level architecture of the system. Each agent will generally be designed to reside on a network node; this agent will monitor the healthiness of a node and perform management functions and recovery tasks. If an agent finds unexpected operation situations, it is allowed to communicate with adjacent agents, which will activate the appropriate recovery strategies. But most management and recovery decisions will be performed locally to prevent the transmission of a large amount of data to the master agent. The next section discusses the researcher's preferred methodology with the developmental model of the system.

3.2. Research Methodology

This research integrates two fields of study namely computer network management and artificial intelligence. There is a great need to investigate how they are used together to achieve network management goal based on intelligent agents. The proposed research follows the Knowledge Engineering (KE) methodology and iterative development model for system implementation. To develop a knowledge-based system, this research follows the methodology described below:

- **Literature Survey -** to acquire and get better understanding of software agents in network management, a detailed literature about the related research was done. This was done through the review of published work, related books and articles; site visits to the SLL were conducted and observations undertaken in an attempt to identify the research problems, motivation, customize the idea and to scale the scope of this research. Detailed literature review on how tradition network management system work and operate was done. The agent languages, platforms and rule engines were reviewed thoroughly to inform the choice of the implementation technologies. A thorough investigation on MAS based network management systems with their methodologies was done as well.

- **Requirements Elicitation -** the system requirements were specified after critically reviewing related research topics and undertaking observations at the SLL site. These requirements include functional and non-functional requirements and are discussed in later sections.

- **Analysis and Design -** the system design was done after analyzing the required technologies and how they will be used together. The literature provided strategies on how to plan and develop a network management tool based software agents.

- **Implementation -** the system was implemented in prototypes with the technologies specified in the literature survey. The technologies used in the implementation of this system are: Java, JADE framework, JESS scripting language, Jpcap library, Hyperic-Sigar library to build an OS independent system. Figure 17 shows the incremental development model followed in the system implementation.

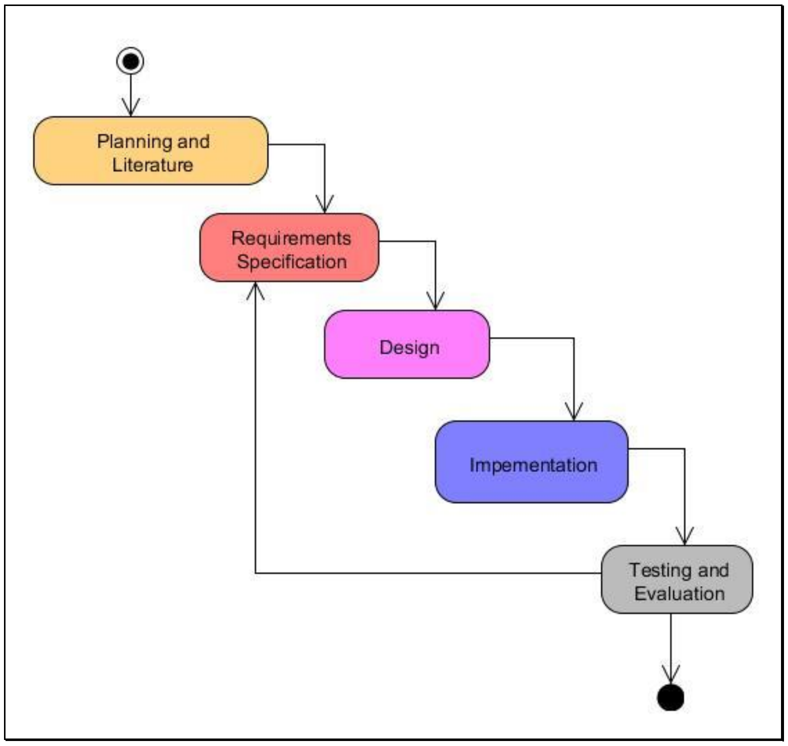


Figure 17: Iterative Development Model for the System Development.

- **Testing and Evaluation** – the implemented system was tested and validated for compliance to the system requirements and for suitability of providing a network management function in marginalized rural areas.

3.3. Agent Development Methodologies

To develop a multi-agent based system, there are standards that have to be followed. This section reviews the current methodologies for the development of an agent-based system. While there are many methods of agent development, they differ in terms of agent theory, language used, suitable rule engine, knowledge acquisition and agent architectures. Another factor that distinguishes these methods is the domain of applicability of the methods. Most researchers no longer develop these methods from the point of inception but find ways to extend the existing methods [59]. The two methodologies that this section will cover are: Extensions of Object-Oriented and Knowledge Engineering Methodologies.

3.3.1. Object-Oriented Methodology

The Object-Oriented Methodology (OOM) is a proven methodology for high-quality object-oriented systems. It involves three stages: 1) specification of system requirements such as functional and non- functional requirements; 2) conversion of the system design into interfaces, classes and method description; and 3) the implementation of a system using object-oriented programming languages such as C++, Java, Eiffel, Python, and Common Lisp Object System (CLOS). The adoption of this methodology would typically end-up in mismatches due to the fact that it uses classes, objects and the client-servers paradigm. One example of OOM is the Agent-Oriented methodology which is the process of explicitly specifying system requirements without any reference to implementation details and a thorough explanation of how the system will achieve the specified requirements [60]. This methodology consists of two stages: 1) The Analysis stage, in which the system developer collects and integrates the system requirements which include, but are not limited to functional requirements, non-function requirements, data requirements, system interface requirements and physical requirements; and 2) the Design phase, in which the system developer designs the system to satisfy the specified system requirements.

Agent-Oriented paradigm employs message passing for communication and can use inheritance and aggregation for defining its design. The advantage of the Agent-Oriented method is the constrained type of messages and the classification of a state in the agent based on its beliefs, desires and intentions [59]. Agent-Oriented methodology introduces the organization of system process like roles, organization, responsibilities, beliefs, desires and intentions. The Agent-Based analysis aims at establishing what the main actors interacting with the system are and how the system interacts with other actors. Further, the analysis seeks to identify what the system is supposed to do. This method associates agents with the system entities according to roles, responsibilities and capabilities with the interactions between them. With the analysis of Agent-Oriented Programming, the system developer has to choose which agents to use and how they interact. There are few Agent-Oriented methodologies and techniques to develop MASs to date and they differ in how they intend to develop the MAS and sometimes they are complementary [61]. Agent-Oriented Methodologies include Gaia, TROPOS, Prometheus, ADELFE, MESSAGE and PASSI methods.

The Gaia methodology allows system developers to easily design a system directly from the defined system requirements. This is the first complete methodology for the analysis and design of MASs and supports the specification of models of the system that are derived from the analysis and design process [62]. The Gaia methodology supports the structure of an agent and the agent environment of execution in the MAS development process. It views the MASs as a system that is made-up of interactive autonomous agents that operate in an organized environment whereby each agent has one or more specific goals. The models defined from the analysis and design process are used to identify roles that agents have to play within the system and the communication protocol between the different roles [63]. Figure 18 shows the relationships between models of Gaia methodology [64]. The Gaia analysis stage consists of role definition that identifies the key roles in the system and interaction model that consists of protocol definition. And the design stage consists of an agent model to identify agent types that will be used in the system under development, service model (input, output, and pre and post condition) to identify services associated with each agent role and acquaintance model for the specification of communication links that exist between agent types [64]. For this reason, Gaia methodology does not deal with the system requirement stage; requirement-capturing stage is considered as an independent of the paradigm used for analysis [61].

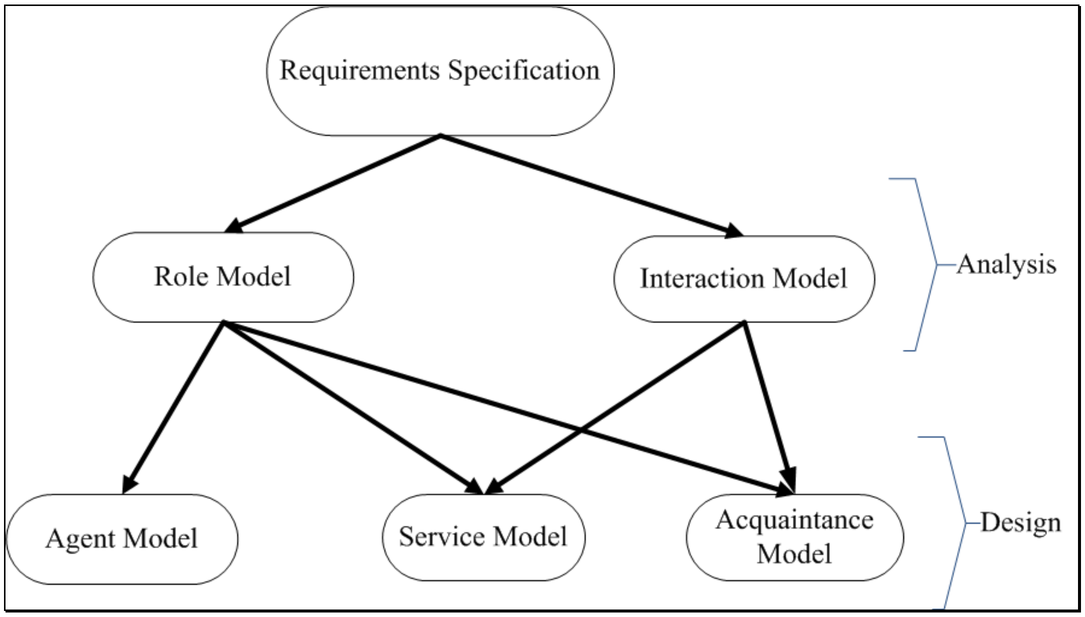


Figure 18: Models of Gaia Methodology [64]**.**

TROPOS is an Agent-Oriented methodology that encompasses the entire software development life cycle. TROPOS is a model-driven methodology that supports belief, desire and intension reasoning mechanism to develop agents. TROPOS, unlike the Gaia methodology, uses a top-down development perspective and supports verification and validation of the development of models and specifications. TROPOS defines two main levels; first, it uses belief, plans and goals reasoning approach throughout the system development life cycle. Second, TROPOS supports the analysis of system requirements in the early stages of the development life cycle, to allow for a thorough understanding of the execution environment of the system. One of the limitations of TROPOS is that it has been fully adopted to develop MASs and lacked the technology that supports the transition between different system development life cycle stages [65].

ADELFE is an Agent-Oriented methodology that uses a top-down development perspective to create adaptive agent architecture. ADELFE guides a system developer in creating adaptive agents through an AMAS theory [66]. The AMAS theory offers a solution to build agents that are going to adapt to their environment of execution. This theory does not support the solution offered by traditional methodologies [66]. ADELFE methodology is not a general methodology like Gaia and TROPOS; it supports systems that are open and complex [67].

Prometheus is an Agent-Oriented methodology that supports iterative development lifecycle through analysis, design and implementation stages. It uses a bottom-up development perspective across system development stages. This methodology supports design of agents that are based on beliefs, intention and require reasoning techniques [68]. System Specification is a stage where the system requirements are explicitly explained using goals (sub-goals) and use case scenarios, Table 2 [68].

Table 2: The Major Models of Prometheus Methodology [68]**.**

Development Stage	**Dynamic Models**	**Structural Overview Models**	**Entity Descriptors**
System Specification	Scenarios	Goals	Functionalities, Actions and percepts
Architectural Design	(Interaction diagrams) Interaction protocols	(Coupling diagrams) (Agent acquaintance) System Overview	Agent Overview

Detailed Design	Process Diagrams	Agent overview Capability overview	Capabilities Plans, Data, Events

Architectural Design is a stage where the system developer states the agent types and captures the overall structure of the system. Thereafter, the system developer defines each agent's role to the overall system by defining its capabilities, data, events and plans in process diagrams [68].

3.3.2. Knowledge Engineering Methodology

The KE Methodology uses an art to acquire knowledge from experts of a specific field to design and develop expert systems [69]. This methodology uses a fundamental technique of interviewing experts, or observing a human/group of experts and study what the experts know and how they reason with their knowledge. The KE methodology includes three key actions through its iterative development life cycle:

- Knowledge Acquisition-allows the expert to enter their knowledge into expert system and allows them to refine later when required [69]. This processed is made of three key stages: knowledge specification, intermediate representation and executable form whereby the intermediate knowledge is presented as rules to the inference engine. Figure 19 shows these stages in system development lifecycle. This is the most challenging stage of KE methodology because a developer has to interview the relevant experts for the proposed system [69].

Knowledge Analysis and Modeling-before the raw data and information that have been captured can be usable; it needs to be transformed to knowledge and this stage deals with that [70].

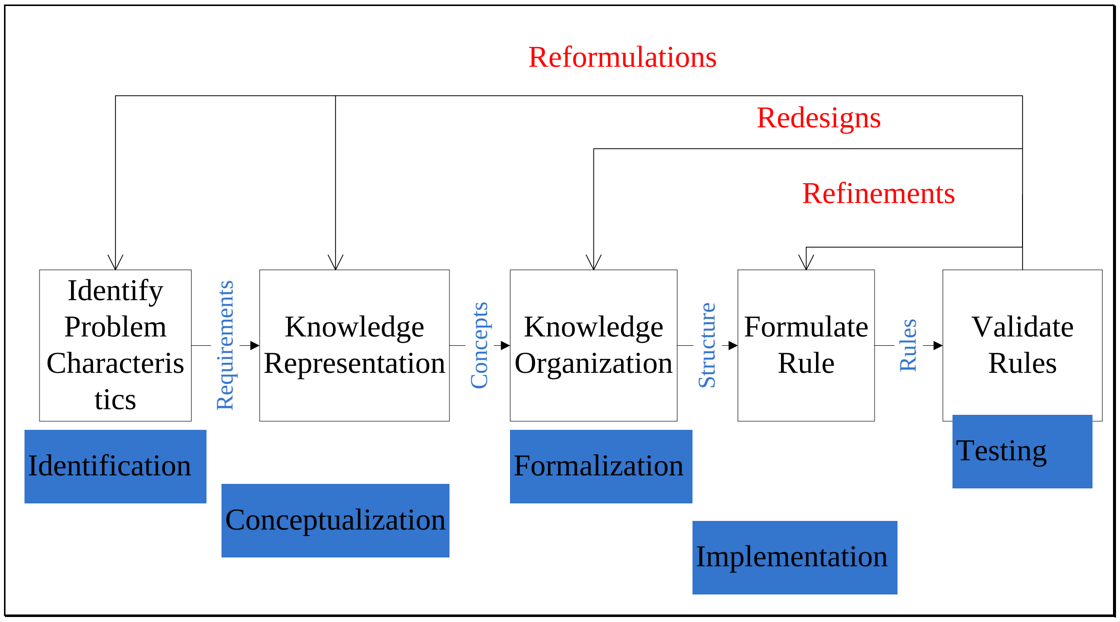


Figure 19: Stages of Knowledge Acquisition (adapted from [69]**).**

- Knowledge Verification-this is the stage at which the developer verifies if the acquired knowledge is for the intended system.

3.4. Requirements Specification

This section elaborates on mandatory and optional requirements of a multi-agent based NMS for rural networks. Some of the general-purpose NMS requirements, such as mapping, auto-discovery, and access control, have been left out of this system, to allow for the development of a lightweight, scalable and flexible system for rural networks.

3.4.1. Functional Requirements

It is universally agreed that these are mandatory functional requirements of a NMS. In this research they have been enhanced to conform to the challenges of rural networks.

- **Fault Management-**the system should be able to detect, analyze and log network problems through monitoring and isolating the problem.

- **Performance Management-** the system has to be built to measure network performance; it has to analyze the normal levels and set/determine appropriate threshold values to ensure high QoS for each service.

- **Configuration of functionality**-NMS should provide support for automatic detection of network nodes (on/off), audit and track interactions with users. This MA based system should support the automatic configuration of new and recovered nodes from fault state. It should support role-based user rights and authenticates against the conventional server. The system should have a database to store all the information of managed devices.

- **Network analysis**-NMS should easily analyze the network data efficiently to avoid any delays, thereby maintaining up-to-date network service.

- **Support of 'disconnected' operation**- the architecture of this NMS should be designed in such a way that management operations are mostly independent of network resources. This eliminates vulnerability when performing network management functions due to link failure or high traffic conditions. This process can be achieved by developing autonomous agent entities, which are capable of performing their decentralized management functions without requiring constant communication with a central manager station. These multiple agent entities should be able to continue their execution even when the communication link with the current agent platform is disrupted or fails by providing a redundant agent platform that will take over when disruption and failures happen on the running agent platform.

- **Interface for State Information**-the design of this system should provide an interface for an administrator. This interface will also allow the administrator to have rights to manually perform management functions. The user interface should be able to show the network topology and show network state information by accessing a specific table/log in the data storage. The NMS should be able to show any alerts when there is a change in the network state and the agents should be able to efficiently react to the changes by providing the required services. This interface will also allow an administrator to easily introduce new services at runtime.

- **Integration with other NMSs**-this system should be easily integrated with other management systems; after all, no management system operates in isolation.

3.4.2. Non-Functional Requirements

- **Usability** - the administrator should be able to use the interface of the NMS without any difficulty, otherwise the quality of service will be degraded in case there is a management function that requires an administrator.

- **Self-recovery**-the autonomous agent entities should be able to self-recover after an unexpected failure of the main-container by using a redundant main-container on another platform. When an agent dies unexpectedly, the adjacent agents, with the same responsibilities, should clone themselves to provide the same management tasks. These MA entities should provide support for persistent so that they will not lose the state of execution when they migrate from node to node.

- **Robust architecture**- the NMS should be able to perform under high abnormal circumstances such as large databases to process and analyze, managing large number of devices, and in case of conflicting rules. It should be able to analyze network data, process it and provide network services.

- **Fault tolerant**-NMS should be able to deal with situations where link or node failures interrupt the normal migration process of roaming MA entities. Fault tolerance features should deal with cases where the node is the manager node itself and ensure that the valuable management information collected by the MA entities is not lost.

- **High performance system** - NMS should be able to foresee possible congestions or failures and take prevention measures before any errors occur. Therefore, it is very important to develop a system infrastructure that guarantees to perform in time demanding factors and provide a sensing tool to prevent any network errors at the same time.

- **Reliable system**-this MA based system should always be reliable to perform network management function like configuration and event management among others.

- **Lightweight footprint** - the system's agents should be designed to be lightweight to the greatest extents possible and provide execution environments that can be installed on any network device, regardless of the node's storage capacity and processing capabilities.

- **Scalable architecture**-scalability is the foremost concern of designing a distributed dynamic system that is going to be deployed in an expanding network. Therefore, the system architecture should be able to provide support for intelligent collectors and processing engines to store aggregated data for a long period. The processing engine should be flexible enough to quickly analyze the collected data to prevent overload and outdated services.

3.5. System Architecture

This section discusses in detail how various software components are to be organized and how they should interact. Section 3.2 explained a methodology this research follows and this section uses that approach to organize the system architecture. The organization of a distributed system is mostly about the software components that constitute the system. This section looks at a decentralized architecture whereby agents are deployed over the network residing on network nodes.

3.5.1. Multi-Service Agent Model

The network management system by delegation is purely based on Multi-Service Agent Model, whereby every agent acts as an active object and has the ability to cooperate and is relatively independent from each other's execution process. Each agent will be designed in a way that it has its own goals and also be capable of coordinating with other agents to reach a global common goal. The agent performance depends on the control and information links between it and other agents. The cooperative performance will be a feature of the system as a whole and the system will be application-independent. This system will not provide high-level cooperation functions; it will only include the cooperative agents according to goals and duties of each application. There are four types of knowledge inside the knowledge base of an agent, which are [71]:

- **Environment knowledge**-partially the agent does not have all the knowledge about its environment, it is only aware of other agents and their purpose. Therefore, it communicates with its adjacent agents to gather information about its environment.

- **Knowledge about itself**- this information consists of skills it possesses, its functions, goals it has to achieve and whether or not it has to collaborate with other agents. In this state, it needs to know what operation it should take when messages arrive at it.

- **Knowledge about problems to be solved**- each agent requires knowing about the general/global problem to be solved.

- **Knowledge about its state**- each agent has to know the processes running in it and whether or not they are working on offered goals or services.

Figure 20 shows the general operation of the Multi-Service Agent Model. This knowledge may be given to the agent upon on startup or it may acquire it dynamically during the course of problem solving. Because agents will be designed on top of JADE platform, the communication amongst them will be facilitated by message passing. JADE platform is a FIPA compliant for the development of agents in Java language.

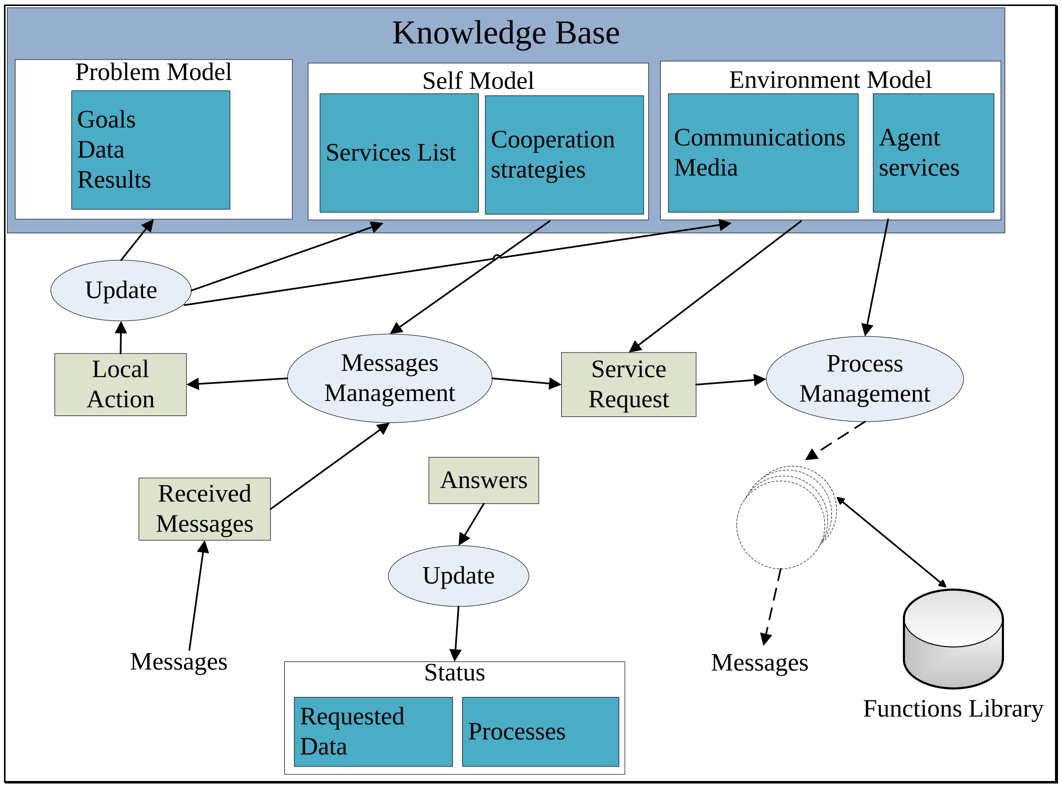


Figure 20: Multi-Service Agent Model (adapted from [71]**).**

3.5.2. System Design

This system is designed as a Multi-Agent environment where each agent performs specific tasks and interacts with other agents. This system consists of *MasterAgent* and *NodeAgent*. These agents are created according to their functions and roles in order to achieve their goals. *MasterAgent* is responsible for visualization of the network state through a web-interface and an agent-platform visualization tool (provided by JADE framework). The *NodeAgent* is responsible for performing the network monitoring and management tasks. This system is designed to have at least two *MasterAgents* that will reside in two separate agent-platforms and a number of *NodeAgents* that will reside in each network node. The creation of two *MasterAgents* is to form a self-recovering and redundant system in case of an agent-platform failure. These agent-platforms are created in such a manner that they possess knowledge about each other's services through their main-container DF. Figure 21 depicts low-level agent architecture; it shows the main functions the *NodeAgent.*

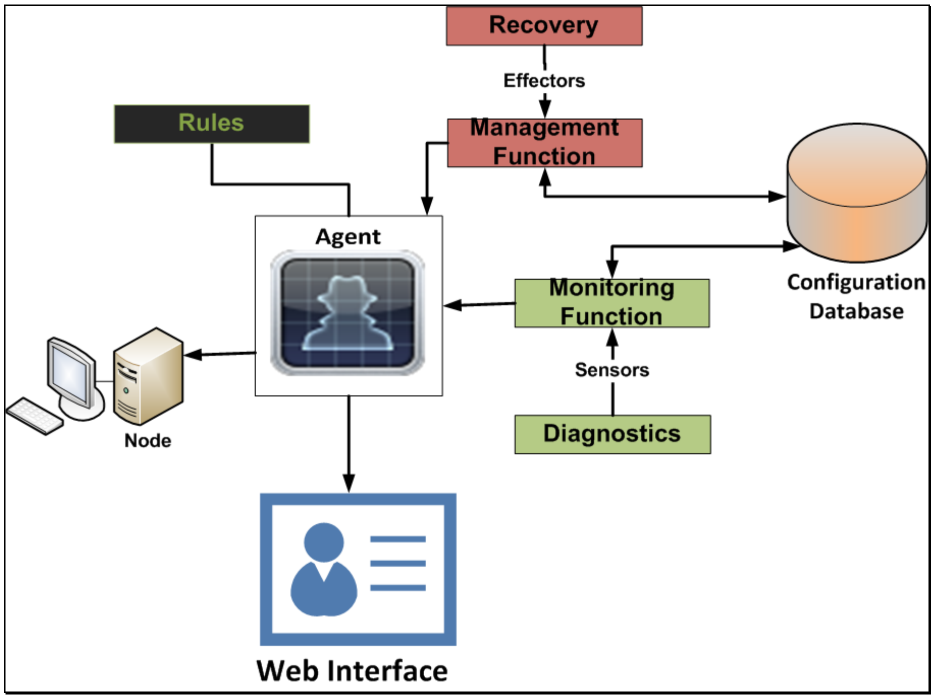


Figure 21: Low-Level Agent Architecture.

The *NodeAgent* main task is to diagnose any network node failures; network anomalies and user operations (access and operation of devices) then execute specified network goals in case of any arising failures/errors. The network goals, rules and policies are specified to the *NodeAgent* through the JESS inference engine. The *NodeAgent* is responsible for the collection of network data and processes it to a usable format then stores it in a database. For the inference engine to perform network diagnosis and fire specified rules, it uses this information stored in a configuration database. The *NodeAgent* ensures the healthiness of network nodes thereby providing high QoS each use requires. The inference engine stores this information as facts in its memory.

Figure 22 below shows a high-level agent operation. Knowledge-base refers to user facts and device status (user data, hypothesis, initial problem data and results) and memory refers to network rules and policies written in Jess, network device is the agents' environment of execution and inference engine is the rule interpreter. The main purpose of an inference engine is to search and select the correct rule to be applied in the agent reasoning process.

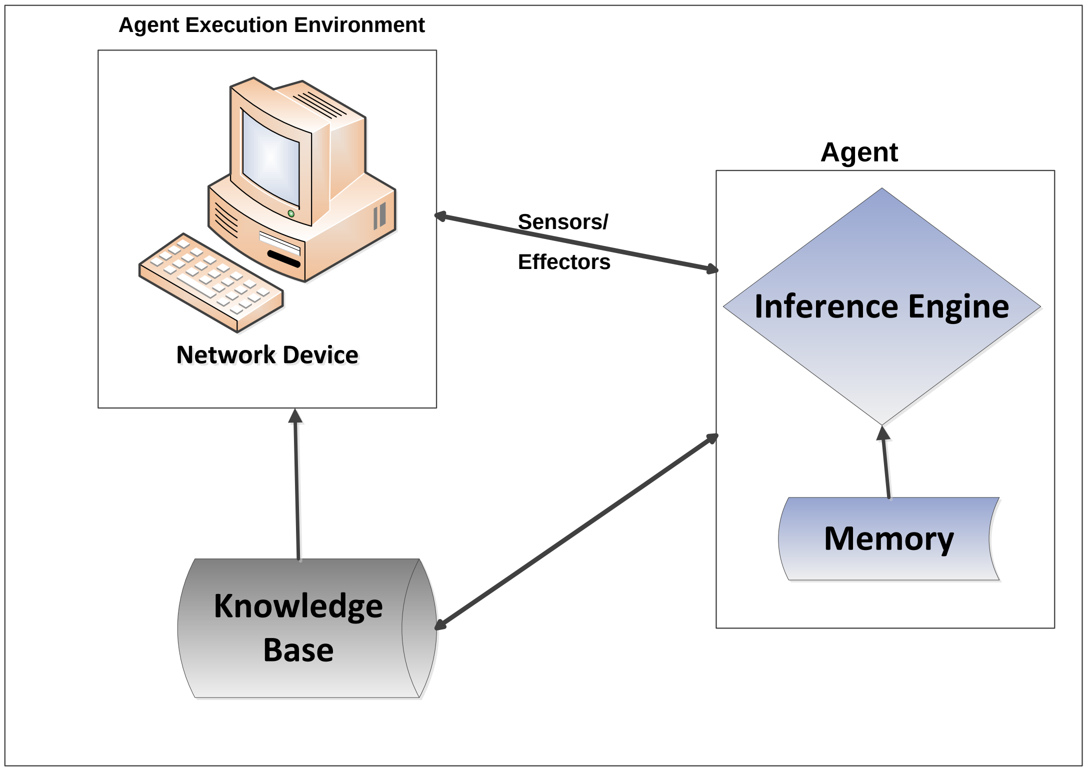


Figure 22: High-Level Agent Operation.

The *MasterAgent* provides a GUI to view the agent-platform and agents registered to it. This option allows the human administrator to start, kill, pause agents, and communicate with other agents as well as communicating with other agent-platforms when required to. Agents communicate with each other using JADE ACL protocol. The overall multi-agent system architecture is presented in Figure 23. Each agent-platform has a main-container that provides AMS, DF, Remote Monitoring Agent (RMA) and our *MasterAgent*. AMS is an agent that has control of the agent-platform and life cycle of agents registered to it. Every agent that is in the agent-platform has to register to the AMS in order to get a valid AID and there can only be one instance of an AMS in one agent-platform. RMA is an agent in the agent-platform that provides support to check agents and agent containers states. DF is an agent that provides services possessed by agents registered to an agent-platform.

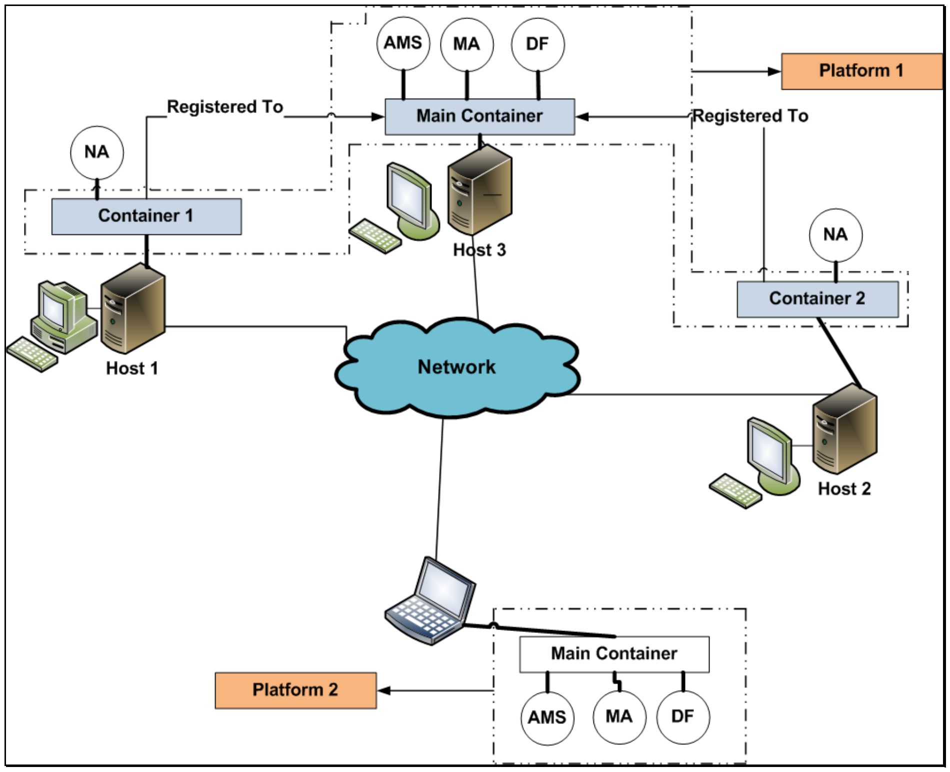


Figure 23: System Architecture.

It is usually called “yellow pages” because it provides visiting agents an opportunity to search the agent-platform for agent’s services and allows agents in the agent-platform to advertise their services. Main-container is a JADE environment that contains agents and containers, there can only be one instance of a main-container in the agent-platform and all other containers must register with it as soon as they start. Platform 2 in Figure 23 is created for redundant and self-recovery system purposes and NA is *NodeAgent* deployed in each network node. Platform 2 has its own main-container as Platform 1 has its main-container.

3.5.3. Agent Description

The system agents are designed in such a way that each agent performs its specific tasks on given network goals. This section gives a detailed view about the operations and functions of the *MasterAgent* and *NodeAgent* involved in the system.

3.5.3.1. Recognition Operation

The recognition operation consists of two key functions, data storage and analysis. The task of data storage is to keep all the data about the possible network breakdown, when it was produced, which agents participate in it, the failure case state, the associated diagnostic operation. The other function is to separate the coming events into different failure cases. The aim of this separation process is to set aside events in sets of events according to problems that caused them. This helps to increase agreement level and reduce computation costs. The computation cost will be reduced if the events are separated into minor groups and associate them separately. The agreement level will be bigger if the resultant set of events is going to diagnostic operation and diagnosis is made as a similar manner. The structure of the separation process is shown in Figure 24.

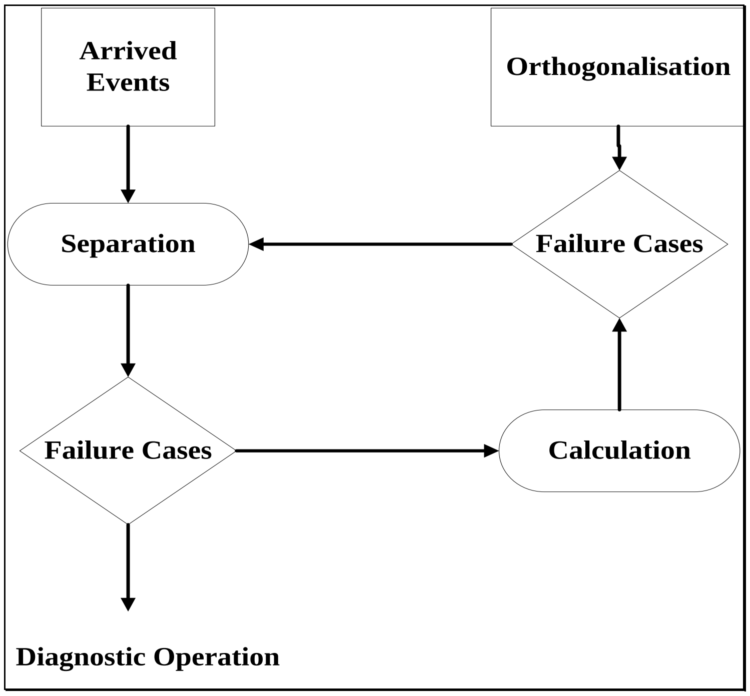


Figure 24: Separation Process (adapted from [71]**).**

When the events arrive at the maintenance system, it is allocated to a set of failure cases or a new one is created. There are four components associated to the network problem; a failure in the network, a set of events caused by it, a set of managed objects affected by the problem and the operations that should be invoked to solve it. Then it makes the separation process to have these steps [71]:

- **Separation-** An event reaches the configuration database and it is allocated to failure case and sent to a diagnostic operation.

- **Calculation-** After the event has been assigned; it is incorporated into one (and only one) failure case. The failure case and structural knowledge on the managed node is used to recalculate the problem scope for the problem. Problem scopes are dynamic and grow as the problem's events are received in the configuration database.

- **Orthogonalisation-** because one event is allocated to one failure case, it is essential that problem possibilities do not intercept, that is, a managed node can only fit to one problem possibility at the same time.

3.5.3.2. Diagnostic Operation

The diagnostic operation is responsible for creating hypotheses concerning the reasons of the network breakdown and then produces a failure reason. The failure cause is received after the symptoms obtained through the recognition operation. These symptoms are collected to structure a failure case. Once the agent has received a list of possible hypotheses, it should prove them using the recovery operation services, thereby generating a single failure cause, which clearly has to be solved. The diagnostic operation has the following elements: the Failure Knowledge which refers to information about the feasible network based on received events; Failure Cases which is the set of events that arrive at the diagnostic operation from the managed node; Structural Knowledge which refers to information about the network node and their operation on the network and Behaviour Knowledge is the information about the production rules. These four components form diagnosis, which is just a set of facts with a high confidence value that are considered a failure/error in the managed node. Before the diagnosis is obtained, while the hypothesis is available, there are four repeated tasks in the diagnosis process [71].

- **Generation of Hypothesis-**this is the step where events and structural knowledge are connected against failure knowledge. In so doing, a set of hypotheses to be validated is obtained.

- **Generation of Questions**-hypotheses and structural knowledge are used to execute backward chaining over production rules to form a set of questions that will be used to validate the hypotheses, through asking the managed node.

- **Network Examination**-after generating questions, the diagnostic operation sends them to recovery operation which will be the one to calculate proof plans on the network and return results to the diagnostic operation. The results are facts with high confidence value.

- **Discrimination**-the results from the examination of the network are used for the forward chaining on the production rules to form difference in confidence values of hypotheses to reach the status of diagnosis.

3.5.3.3. Recovery Operation

The primary aim of this function is to determine the operations required for the agents in different stages of management process and order their execution. Recover operation is responsible for the set of strategies explaining the modifications that must be provoked in the state of the managed node and can request agents to change attribute values of managed node. Here are the key set of services that need to be carried out, the generation of plans and execution and retrieving results [71]:

- **Repair Plans**-once the diagnostic operation has found any failures through recognition operation, recovery operation is notified about the legitimate failure cause. This operation will make a repair plan in order to resolve the failure. The recovery operation will try to find the best possible solution and the actions that lead to this failure. After making a solution, it will run the solution and notify the recognition operation about the results.

- **Reporting and Notification Plans**-when there is a network failure cause, the recovery operation keeps log about the failures and report them to a *MasterAgent*. These logs should be presented in the NMS interface and be kept for future purposes, for example; agents should know about the frequent failures, in so doing, it will make it easier for the recovery operation to know what to do when the same problem arises without even going through the process of generating a new solution.

- **Validation Plans**-this plan is used to make sure that the network breakdown has been suitably fixed. It uses logs of saved by the recovery operation to the *MasterAgent* to acquire knowledge about the state of knowledge about the managed node, the agents running on the system and if a solution state has been achieved.

3.5.3.4. System Management Operation

The network administrator uses this operation to get the full control of the network management. The administrator uses the system management operation to monitor any function of the system, such as, incoming alarms, diagnosis processes, repairing processes that define the overall state of the system and can also interact with any of the agents in the system.

3.5.4. System Operation

This section models the dynamic aspect of the system when it is operating. This section will present use case, sequence diagram and activity diagram. The use case diagram will exhibit the system functionality using system requirements information. The activity diagram shows message flow from one activity to another in the system and the sequence diagram captures the time sequence of message flow from one object to another.

3.5.4.1. Use Case Diagram

Figure 25 shows the interaction of the system with the administrator and the device user. The device user is only allowed to utilize the network node while the network administrator monitors and manages the system agents and their operation.

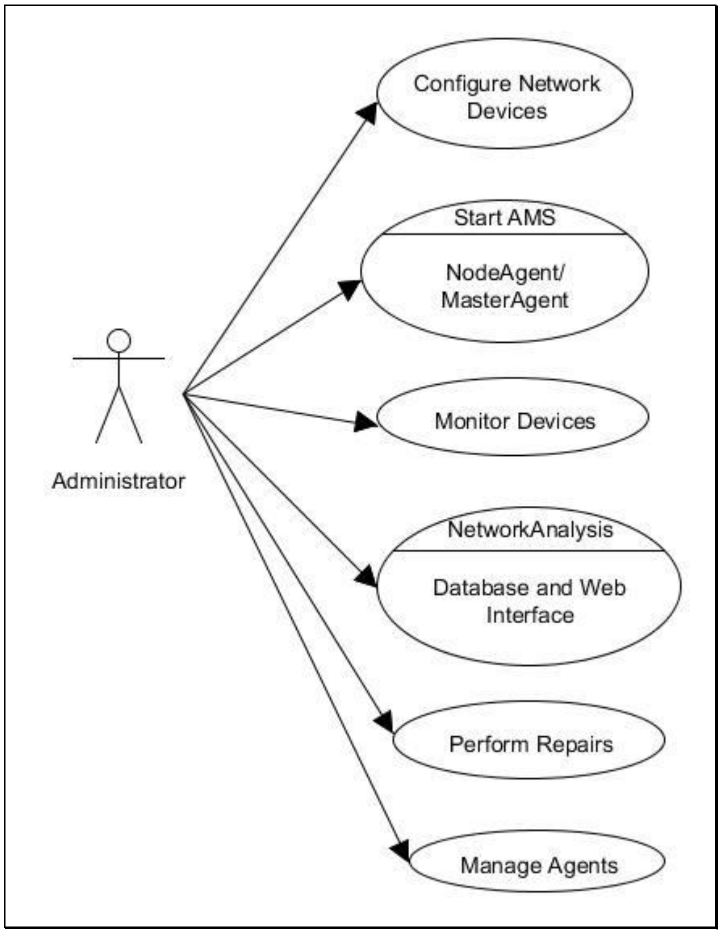


Figure 25: Use case Diagram.

3.5.4.2. Sequence Diagram

Figure 26 shows the objects (Administrator, *MasterAgent*, *NodeAgent* and managed Device) taking part in the interaction, message flow amongst objects, the sequence in which messages are flowing and the object organization.

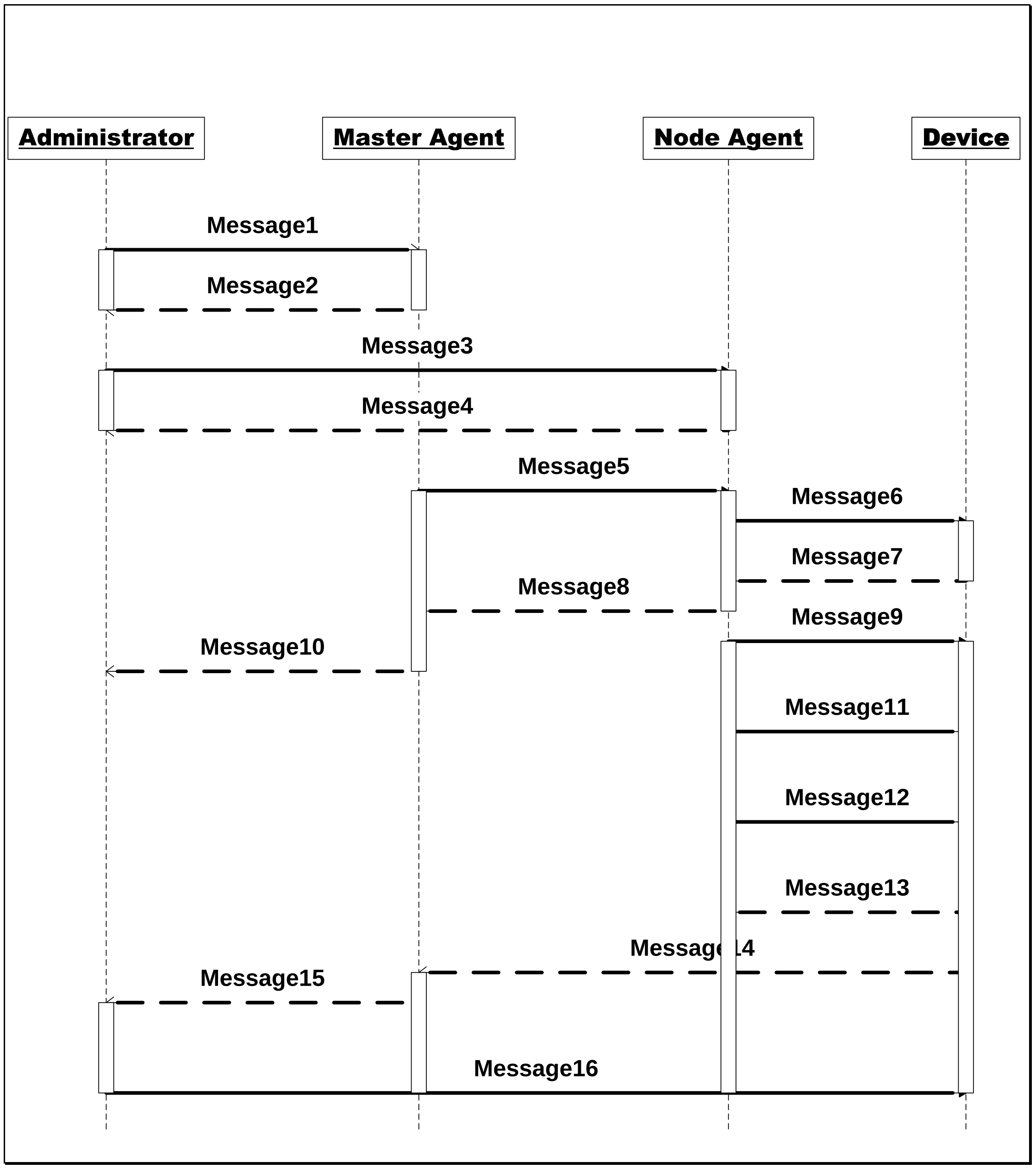


Figure 26: Sequence Diagram.

Message 1: the administrator starts the *MasterAgent* thereby starting the AMS *main-container*.

Message 2: the *MasterAgent* registers its services to the AMS and creates the agent-platform environment.

Message 3: the administrator starts the *NodeAgent*.

Message 4: the *NodeAgent* starts its container and registers its services to the AMS.

Message 5: once the *NodeAgent* is started, the *MasterAgent* monitors its state and keeps on sending ‘still alive’ messages.

Message 6: when the *NodeAgent* is instantiated it first gets the state of the executing environment, the *NodeAgent* retrieves the user, hardware and network state.

Message 7: the managed device responds with the required information.

Message 8: the *NodeAgent* sends the device state to the *MasterAgent*.

Message 9: the NodeAgent consistently monitors the device.

Message 10: the *MasterAgent* updates the administrator database thereby updating the visualization web interface with the network statistics and user information.

Message 11: the *NodeAgent* performs basic network management with its database as a knowledge base and current events as an agent agenda.

Message 12: the *NodeAgent* waits for the triggers and alarm should there be network rules fired.

Message 13: should there be thresholds reached and rules fired *NodeAgent* receives alarms.

Message 14: the NodeAgent notifies the MasterAgent about the occurred events.

Message 15: then the *MasterAgent* notifies the administrator and this is also shown on the web interface.

Message 16: the administrator is able to perform management functions based on the received alarms.

3.6. Conclusion

This chapter presented the design of a network management based on multi-agents architecture. This system has agents that will perform network management by delegation; each agent is designed to perform its specific task when required to and also to collaborate with other agents when the situation demands, in order to achieve a common goal. The next chapter discusses the implementation of the proposed design.

4. Implementation

This chapter describes the development process of a multi-agent based network management system using Java, JESS and MySQL as programming languages. The development process of this system follows the system design and requirements specified in Chapter 3.

4.1. Introduction

This chapter summarizes the implementation details of the *NodeAgent*, *MasterAgent*, database storage and the web interface. Where applicable, source code or interfaces are used to show how the modules are implemented. Section 4.2 and 4.3 present the implementation of the *MasterAgent* and the *NodeAgent*, respectively. Section 4.4 describes the implementation of an inference engine and Section 4.5 shows how to start the agent platform with the specification of system agents' communication method implementation and the creation a decentralized system to form a Multi-Agent System

The system is implemented on Microsoft Windows, Linux Ubuntu and Mac OS to ensure platform independence.

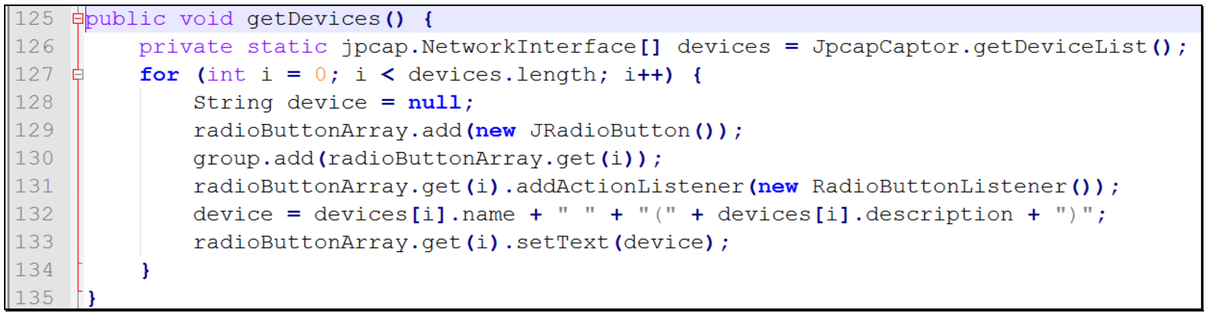
4.2. *MasterAgent*

The *MasterAgent* is responsible for creating an agent-execution environment and the agent-platform, and for starting the AMS. *MasterAgent* consists of: a bandwidth meter; a network statistics module (that monitors the bandwidth utilization); the AMS interface, simple web interface developed in Java Servlets; as well as data storage to store all the information about the managed devices. All the data presented in these interfaces is received from the *NodeAgent* in set time periods to help the administrator to understand the behaviour of the managed devices. This section discusses the Java classes and agent behaviours used to implement the agent components.

The process of network management starts with monitoring network resources such as bandwidth usage, sites visited and applications running on user machines. This section describes how the network statistics are collected, analyzed and presented. Following is a step-by-step implementation of the *MasterAgent*.

4.2.1. Bandwidth Meter Behaviour

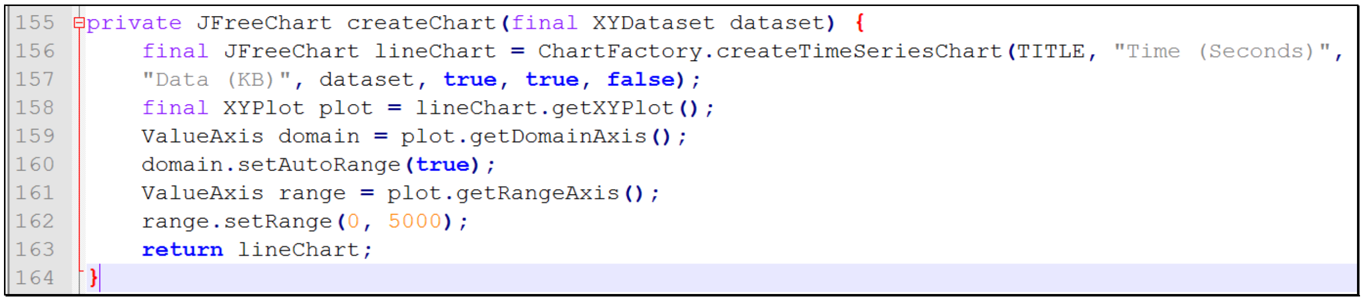
This behavior is responsible for the presentation of bandwidth utilization, it uses Jpcap library to get all the network interfaces of a machine and the JFreeChart library to draw a dynamic chart that present bandwidth (KB/S) usage. The GUI lists the network interfaces and the administrator is allowed to select the one they want and start to monitor. We use the *getDevice()* method to retrieve the network interface and this method is implemented as follows:



```
public void getDevices() {
    private static jpcap.NetworkInterface[] devices = JpcapCaptor.getDeviceList();
    for (int i = 0; i < devices.length; i++) {
        String device = null;
        radioButtonArray.add(new JRadioButton());
        group.add(radioButtonArray.get(i));
        radioButtonArray.get(i).addActionListener(new RadioButtonListener());
        device = devices[i].name + " " + "(" + devices[i].description + ")";
        radioButtonArray.get(i).setText(device);
    }
}
```

Figure 27: Method to List Network Interfaces.

Figure 27 shows a code extract that retrieves the physical network interfaces and adds them to a grouped radio button array (to allow an administrator to choose only one option at a time) and then lists them in a GUI. The GUI has a start button that implements an action listener that starts the process of capturing real-time bandwidth usage. The real-time bandwidth capturing is facilitated by swing worker method that allows for the execution of a long-running GUI task. Figure 28 shows a code extract that implements a line chart.



```
155 private JFreeChart createChart(final XYDataset dataset) {
156     final JFreeChart lineChart = ChartFactory.createTimeSeriesChart(TITLE, "Time (Seconds)",
157     "Data (KB)", dataset, true, true, false);
158     final XYPlot plot = lineChart.getXYPlot();
159     ValueAxis domain = plot.getDomainAxis();
160     domain.setAutoRange(true);
161     ValueAxis range = plot.getRangeAxis();
162     range.setRange(0, 5000);
163     return lineChart;
164 }
```

Figure 28: Method to Create Line Chart (KB/S).

JFreeChart Time Series method allows us to plot set of consumed data values over a time period, the third argument of this method, line 157, is a set of values (data/time) captured in real-time network traffic. The bandwidth meter behaviour implements the JADE simple behaviour that allows it to run once when the agent starts. This helps an administrator to watch network traffic whenever they want by just starting the GUI thread. Figure 29 shows the bandwidth monitor interface. The interface has the *Start Monitor* button, list of active network interfaces and the line graph view that shows consumed data in kilobytes over time in seconds.

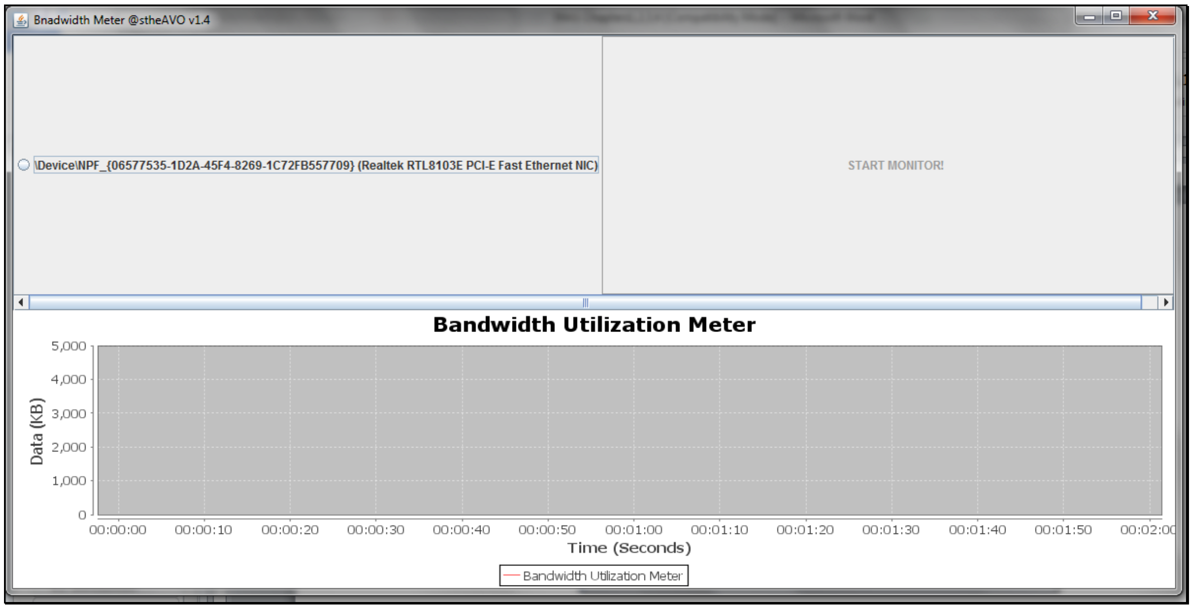
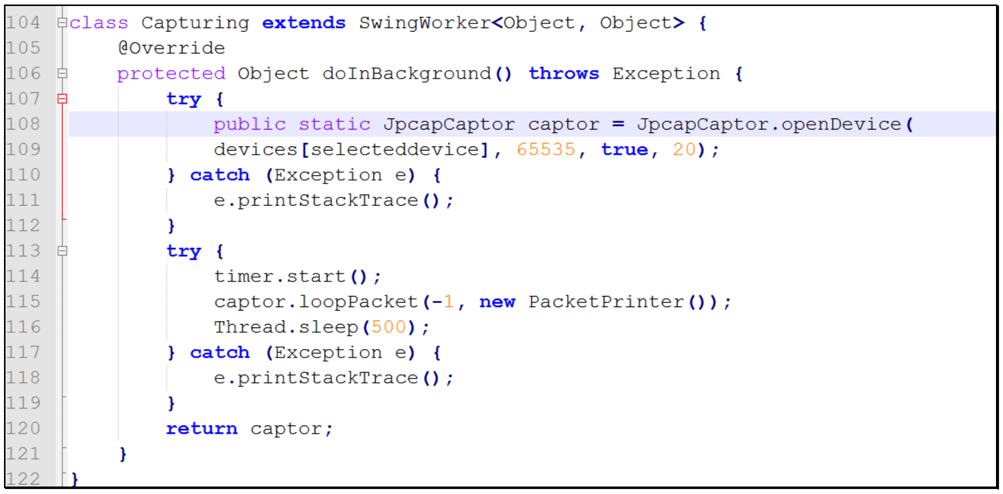

Bnadwidth Meter @stheAVO v1.4
\Device\NPF_{06577535-1D2A-45F4-8269-1C72FB557709} (Realtek RTL8103E PCI-E Fast Ethernet NIC)
START MONITOR!
Bandwidth Utilization Meter
Data (KB)
5,000
4,000
3,000
2,000
1,000
0
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:50
00:01:00
00:01:10
00:01:20
00:01:30
00:01:40
00:01:50
00:02:00
Time (Seconds)
Bandwidth Utilization Meter

Figure 29: Bandwidth Monitor Interface.

Figure 30 shows the implementation of a swing worker thread that runs forever with a break of 500 milliseconds (0.5 seconds). The capturing method makes use of the selected network interface, line 108 & 109, and when the thread timer starts, line 114, it captures the network traffic, line 115, indefinitely and loop the data to the *PacketPrinter()* class. *PacketPrinter()* class is responsible for printing the network traffic contents to the bandwidth meter GUI.

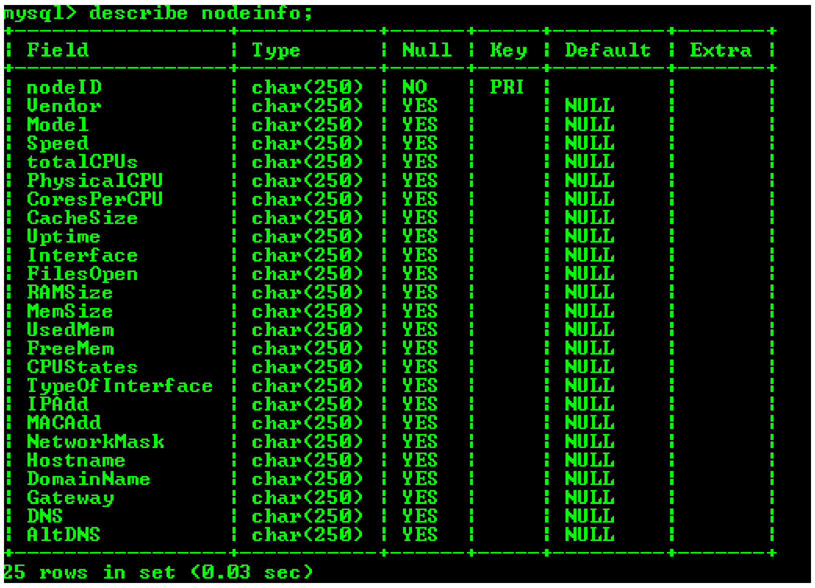


```
104 class Capturing extends SwingWorker<Object, Object> {
105     @Override
106     protected Object doInBackground() throws Exception {
107         try {
108             public static JpcapCaptor captor = JpcapCaptor.openDevice(
109             devices[selecteddevice], 65535, true, 20);
110         } catch (Exception e) {
111             e.printStackTrace();
112         }
113         try {
114             timer.start();
115             captor.loopPacket(-1, new PacketPrinter());
116             Thread.sleep(500);
117         } catch (Exception e) {
118             e.printStackTrace();
119         }
120         return captor;
121     }
122 }
```

Figure 30: Method to Show Data Capturing Thread.

4.2.2. System and User Info Behaviour

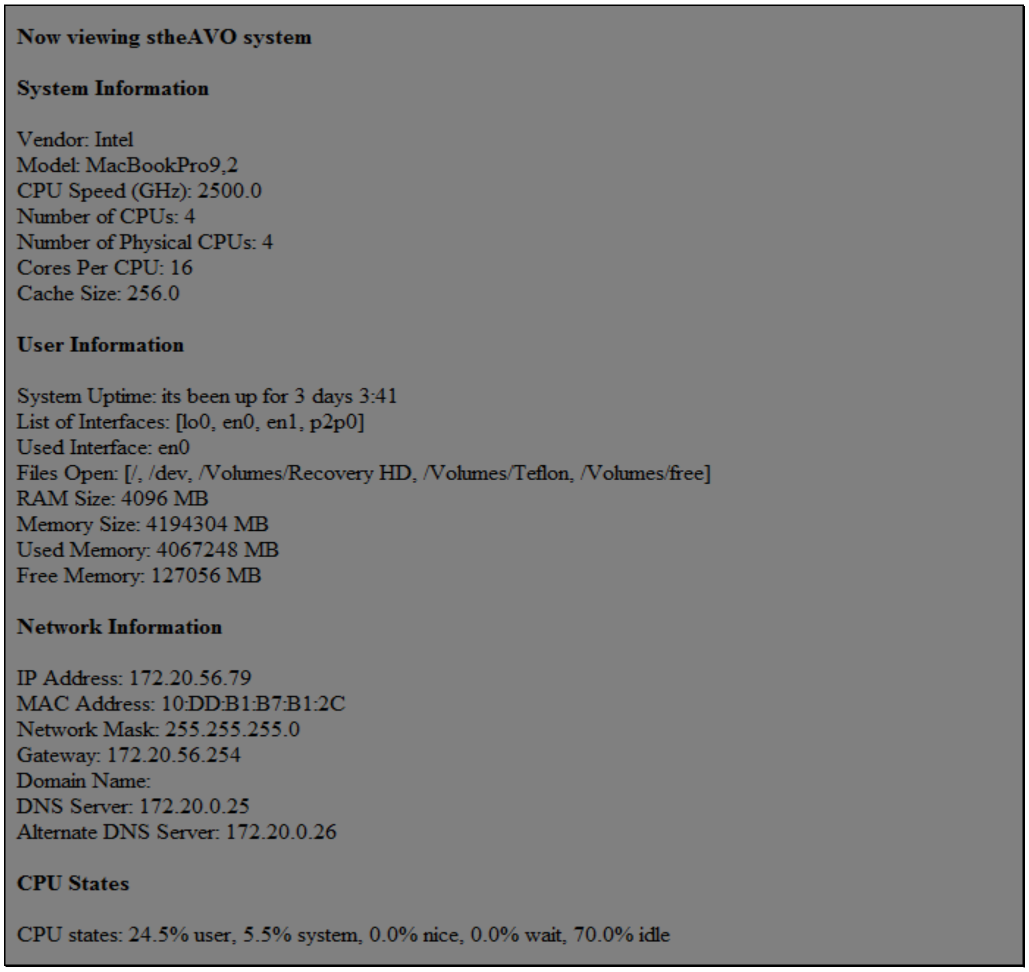
The user information is used to associate the consumed network resources with the current user of a network device. And the system information is used to gather hardware and software information of the devices in the network. The *NodeAgent* sends this data to the *MasterAgent* for the administrator to learn about running OSs, softwares installed on managed devices and applications running on managed devices, thereby allowing the administrator to perform management functions. The managed device network information helps to plot the network topography for example IP addresses can be used to learn about devices connected to the Internet and detect if there is a problem with the network interface or the specific router. Figure 31 shows the important aspects of the system and the user information. The system acquires from a managed device to a database table *NodeInfo*.



```
mysql> describe nodeinfo;
+-------------------+-----------+------+-----+---------+-------+
| Field             | Type      | Null | Key | Default | Extra |
+-------------------+-----------+------+-----+---------+-------+
| nodeID            | char(250) | NO   | PRI |         |       |
| Vendor            | char(250) | YES  |     | NULL    |       |
| Model             | char(250) | YES  |     | NULL    |       |
| Speed             | char(250) | YES  |     | NULL    |       |
| totalCPUs         | char(250) | YES  |     | NULL    |       |
| PhysicalCPU       | char(250) | YES  |     | NULL    |       |
| CoresPerCPU       | char(250) | YES  |     | NULL    |       |
| CacheSize         | char(250) | YES  |     | NULL    |       |
| Uptime            | char(250) | YES  |     | NULL    |       |
| Interface         | char(250) | YES  |     | NULL    |       |
| FilesOpen         | char(250) | YES  |     | NULL    |       |
| RAMSize           | char(250) | YES  |     | NULL    |       |
| MemSize           | char(250) | YES  |     | NULL    |       |
| UsedMem           | char(250) | YES  |     | NULL    |       |
| FreeMem           | char(250) | YES  |     | NULL    |       |
| CPUStates         | char(250) | YES  |     | NULL    |       |
| TypeOfInterface   | char(250) | YES  |     | NULL    |       |
| IPAdd             | char(250) | YES  |     | NULL    |       |
| MACAdd            | char(250) | YES  |     | NULL    |       |
| NetworkMask       | char(250) | YES  |     | NULL    |       |
| Hostname          | char(250) | YES  |     | NULL    |       |
| DomainName        | char(250) | YES  |     | NULL    |       |
| Gateway           | char(250) | YES  |     | NULL    |       |
| DNS               | char(250) | YES  |     | NULL    |       |
| AltDNS            | char(250) | YES  |     | NULL    |       |
+-------------------+-----------+------+-----+---------+-------+
25 rows in set (0.03 sec)
```

Figure 31: NodeInfo Table Fields.

The information stored in table *NodeInfo* is presented to a web interface implemented on Java servlets. The information is updated in the data storage using the JADE simple behaviour and an administrator has an option to choose from the web GUI which device they want to check system specs. Figure 32 shows the information of *stheAVO* device; the web-interface shows the CPU states, network, system and user information.

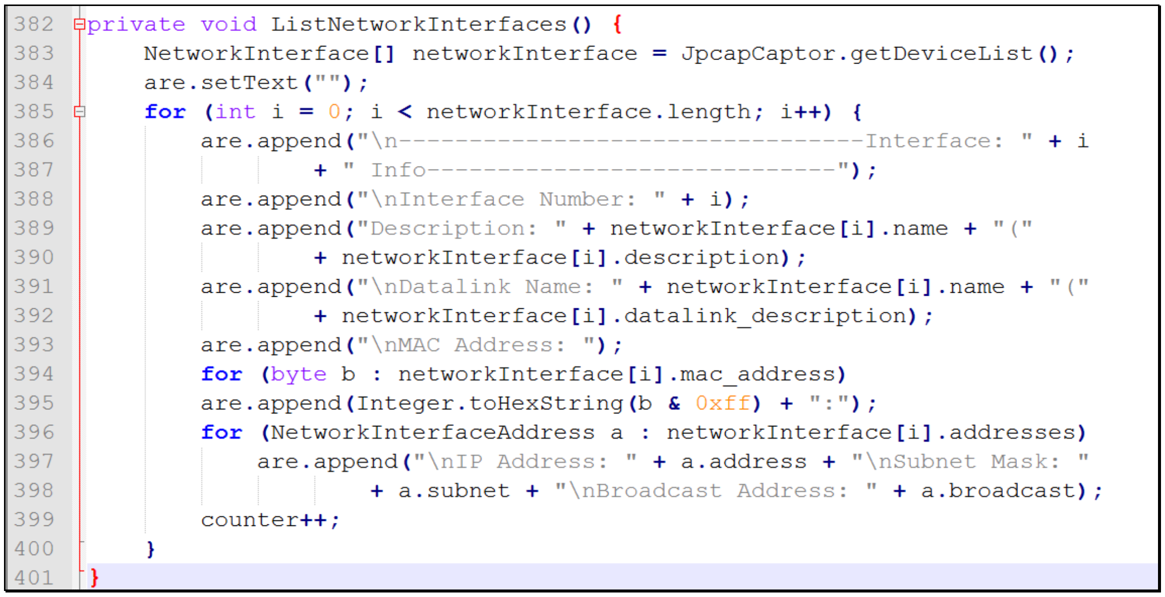


```
Now viewing stheAVO system

System Information

Vendor: Intel
Model: MacBookPro9,2
CPU Speed (GHz): 2500.0
Number of CPUs: 4
Number of Physical CPUs: 4
Cores Per CPU: 16
Cache Size: 256.0

User Information

System Uptime: its been up for 3 days 3:41
List of Interfaces: [lo0, en0, en1, p2p0]
Used Interface: en0
Files Open: [/, /dev, /Volumes/Recovery HD, /Volumes/Teflon, /Volumes/free]
RAM Size: 4096 MB
Memory Size: 4194304 MB
Used Memory: 4067248 MB
Free Memory: 127056 MB

Network Information

IP Address: 172.20.56.79
MAC Address: 10:DD:B1:B7:B1:2C
Network Mask: 255.255.255.0
Gateway: 172.20.56.254
Domain Name:
DNS Server: 172.20.0.25
Alternate DNS Server: 172.20.0.26

CPU States

CPU states: 24.5% user, 5.5% system, 0.0% nice, 0.0% wait, 70.0% idle
```

Figure 32: Web Interface to Show System and User Info.

4.2.3. Packet Capture Behaviour

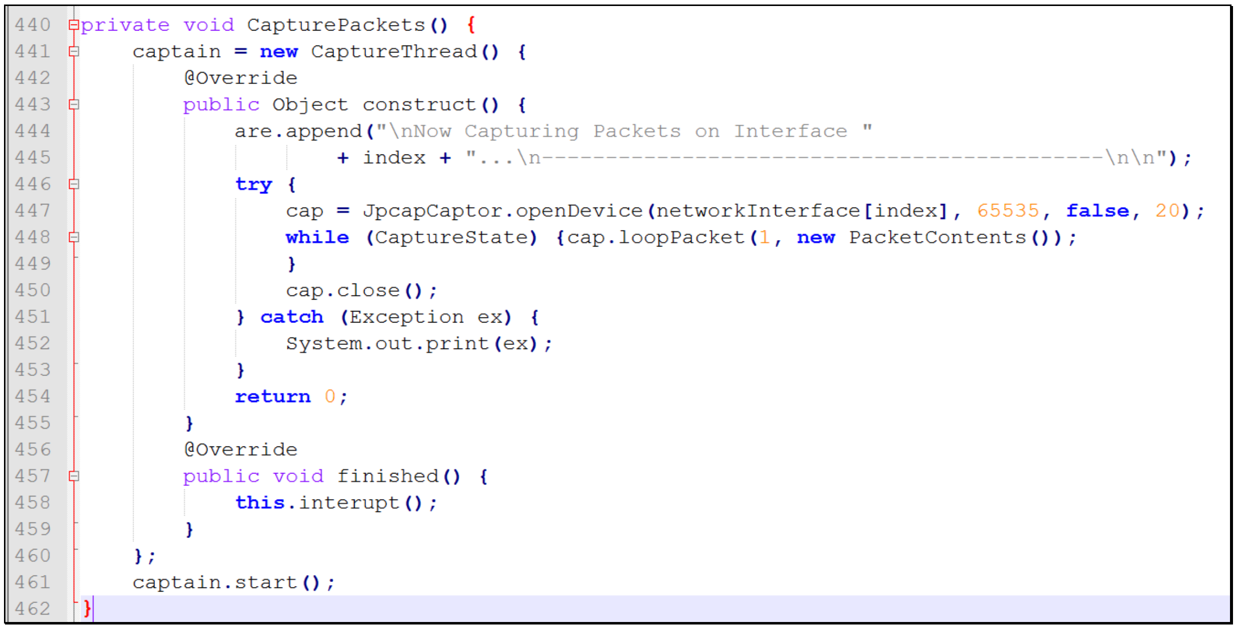
This behavior is responsible for capturing network packets in real time for the selected network interface and this process makes use of Jpcap library. The network capturing interface allows a user to list all network interfaces, filter network packets according to their Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) port numbers, save and load network packets to a file for packet analysis. On Figure 33 the *ListNetworkInterfaces()* method lists device network interfaces and their details. These details include broadcast, MAC and IP address, interface number and description, data link name and subnet mask.



```
382 private void ListNetworkInterfaces() {
383     NetworkInterface[] networkInterface = JpcapCaptor.getDeviceList();
384     are.setText("");
385     for (int i = 0; i < networkInterface.length; i++) {
386         are.append("\n---------------------------------Interface: " + i
387                 + " Info-----------------------------");
388         are.append("\nInterface Number: " + i);
389         are.append("Description: " + networkInterface[i].name + "("
390                 + networkInterface[i].description);
391         are.append("\nDatalink Name: " + networkInterface[i].name + "("
392                 + networkInterface[i].datalink_description);
393         are.append("\nMAC Address: ");
394         for (byte b : networkInterface[i].mac_address)
395         are.append(Integer.toHexString(b & 0xff) + ":");
396         for (NetworkInterfaceAddress a : networkInterface[i].addresses)
397             are.append("\nIP Address: " + a.address + "\nSubnet Mask: "
398                     + a.subnet + "\nBroadcast Address: " + a.broadcast);
399         counter++;
400     }
401 }
```

Figure 33: Method to Show Network Interface Details.

The MAC address is retrieved as a byte, line 394, and converted to a hexadecimal string. C*ounter++*, line 399, ensures that all the network interfaces are written on the text area in the order of their details. Figure 34 shows the *CapturePackets()* method that takes the selected network interface and capture network packets in real time.



```
440 private void CapturePackets() {
441     captain = new CaptureThread() {
442         @Override
443         public Object construct() {
444             are.append("\nNow Capturing Packets on Interface "
445                     + index + "...\n--------------------------------------------\n\n");
446             try {
447                 cap = JpcapCaptor.openDevice(networkInterface[index], 65535, false, 20);
448                 while (CaptureState) {cap.loopPacket(1, new PacketContents());
449                 }
450                 cap.close();
451             } catch (Exception ex) {
452                 System.out.print(ex);
453             }
454             return 0;
455         }
456         @Override
457         public void finished() {
458             this.interupt();
459         }
460     };
461     captain.start();
462 }
```

Figure 34: Method to Capture Network Packets.

The *CapturePackets()* method first initializes the *CaptureThread()* class, line 441, that facilitates the real time capturing process. This thread ensures that a user can easily start and stop the capturing process without the process freezing. The *CapturePackets()* method overrides the *construct()* and *finish()*abstract methods. The *construct()* method implements the *Runnable()* interface to run the thread indefinitely till the capturing process is stopped. Line 447 opens the selected network interface (*index*) and line 448 captures packets as long as the thread state is true (running). Figure 35 shows the packet capture interface with radio buttons used to filter network packets according to their TCP and UDP ports, the interface also has an option to save packets and load them later for analysis.

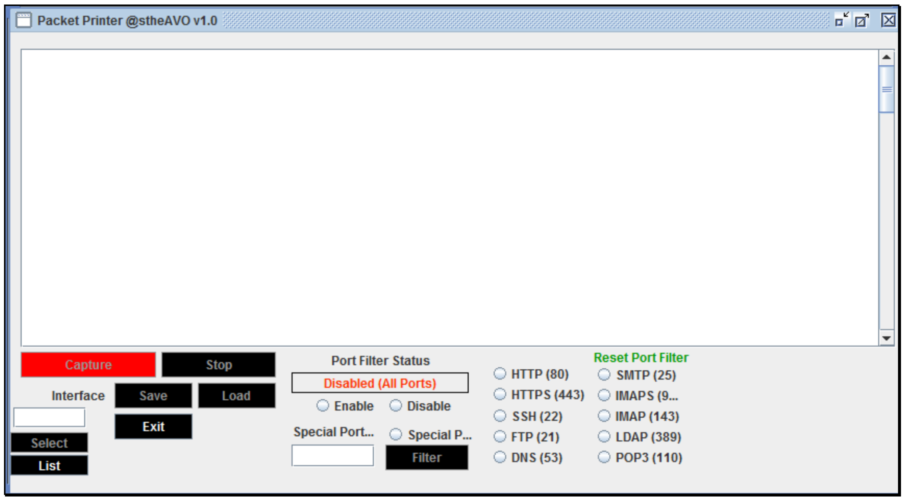


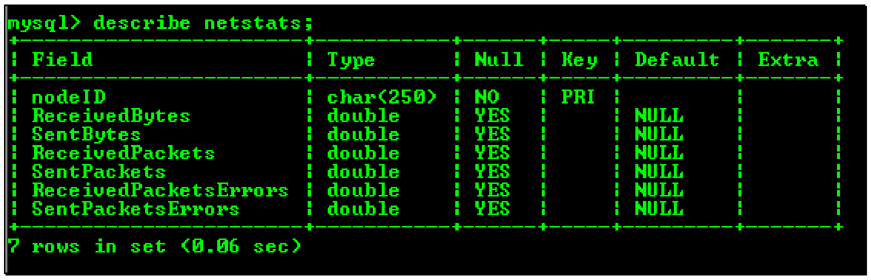
Figure 35: Packet Capture Interface.

At start the interface shows inactive *Capture*, *Stop*, *Select*, *Save*, *Load*, and *Filter* buttons; because at start a user has to choose which interface they want to sniff by listing (*List* button) the active network interface. When a user clicks on *Exit* button the GUI will close but not stopping the *MasterAgent*.

For management purposes, this behaviour sniffs network packets to check who is communicating with whom, type of communication, packet contents and TCP/IP protocols used. This information helps the administrator to block the untrusted device and easily detect network intrusion (this is beyond the scope of this research) by learning the source and destination IP addresses. An administrator has an option to either analyze network packets in real-time or in an offline mode (where packets are loaded from a file log).

4.2.4. Network Statistics Behaviour

The *MasterAgent* collects this data from *NodeAgents* in the network to graph network statistics and compare the data usage of managed devices. This behaviour is also responsible for archiving the consumed data and network packets of a week period of time of the whole network as the *NodeAgent* is doing the same for the each device. Figure 36 describes the device network statistic information of interest. This information is stored in *Netstats* table and is used to manage the data usage. The *nodeID* field is the name of the user logged-in in the managed device. The *nodeID* field is used to associate the data consumed with current user of the device. The packet error fields (*ReceivedPacketsErrors* and *SentPacketsErrors)* are used to check the number of packet errors the user experiences.



```
mysql> describe netstats;
+-----------------------+-----------+------+-----+---------+-------+
| Field                 | Type      | Null | Key | Default | Extra |
+-----------------------+-----------+------+-----+---------+-------+
| nodeID                | char(250) | NO   | PRI |         |       |
| ReceivedBytes         | double    | YES  |     | NULL    |       |
| SentBytes             | double    | YES  |     | NULL    |       |
| ReceivedPackets       | double    | YES  |     | NULL    |       |
| SentPackets           | double    | YES  |     | NULL    |       |
| ReceivedPacketsErrors | double    | YES  |     | NULL    |       |
| SentPacketsErrors     | double    | YES  |     | NULL    |       |
+-----------------------+-----------+------+-----+---------+-------+
7 rows in set (0.06 sec)
```

Figure 36: Fields of *Netstats* Database Table.

Figure 37 shows a dynamic JFreeChart bar chart for the comparison of data consumed and network packets over the system uptime of the managed device. The *MasterAgent* retrieves the data from its database and presents it graphically.

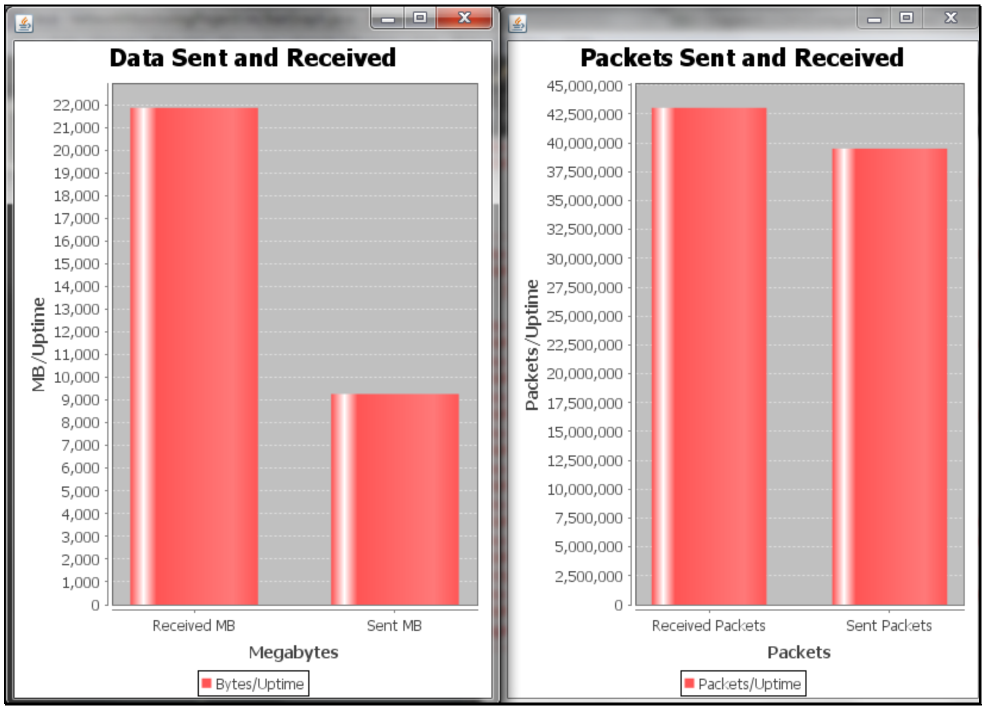


Figure 37: Data Consumed and Packets Captured in a Period of System Uptime.

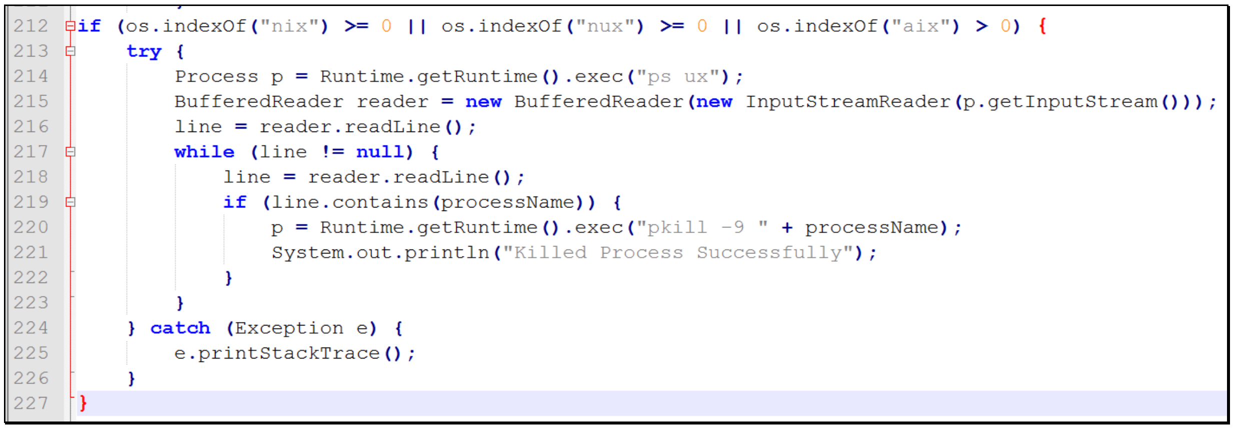
4.3. *NodeAgent*

The role of the *NodeAgent* is to fully reside on a network device and perform network monitoring and basic management functions and send user and device information to the *MasterAgent*. This agent, like the *MasterAgent*, has a built-in inference engine developed in Jess (explained later in this chapter). When the *NodeAgent* is started it registers services it can offer and publishes them to the DF. When *NodeAgent* dies, it deregisters from the DF. This section discusses different behaviours of the *NodeAgent*, which include data collection, analysis and basic management.

4.3.1. System and User Info Behaviour

This behaviour uses Hyperic-Sigar library to gather the user and system information to the MySQL database table *NodeInfo*. The information of interest includes but is not limited to system uptime, CPU time, used network interface, device vendor and model, device IP and MAC address, network gateway, device operating system and device memory among others. This behaviour is also responsible for the collection of device running applications (task manager) to sniff which applications the user is running that might be not acceptable to the network. The agent's inference engine uses this information to apply the given rules. For example, if a user is running a torrent download manager, the agent can detect that the user is trying to download a big file and kill that process.

Figure 38 shows a snippet of a *killProcess()* method that kills a detected process, the method first checks the running OS and analyzes it accordingly. The agent keeps on reading the system runtime applications into an input buffer reader and analyzes them line-by-line.



```
212 if (os.indexOf("nix") >= 0 || os.indexOf("nux") >= 0 || os.indexOf("aix") > 0) {
213     try {
214         Process p = Runtime.getRuntime().exec("ps ux");
215         BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));
216         line = reader.readLine();
217         while (line != null) {
218             line = reader.readLine();
219             if (line.contains(processName)) {
220                 p = Runtime.getRuntime().exec("pkill -9 " + processName);
221                 System.out.println("Killed Process Successfully");
222             }
223         }
224     } catch (Exception e) {
225         e.printStackTrace();
226     }
227 }
```

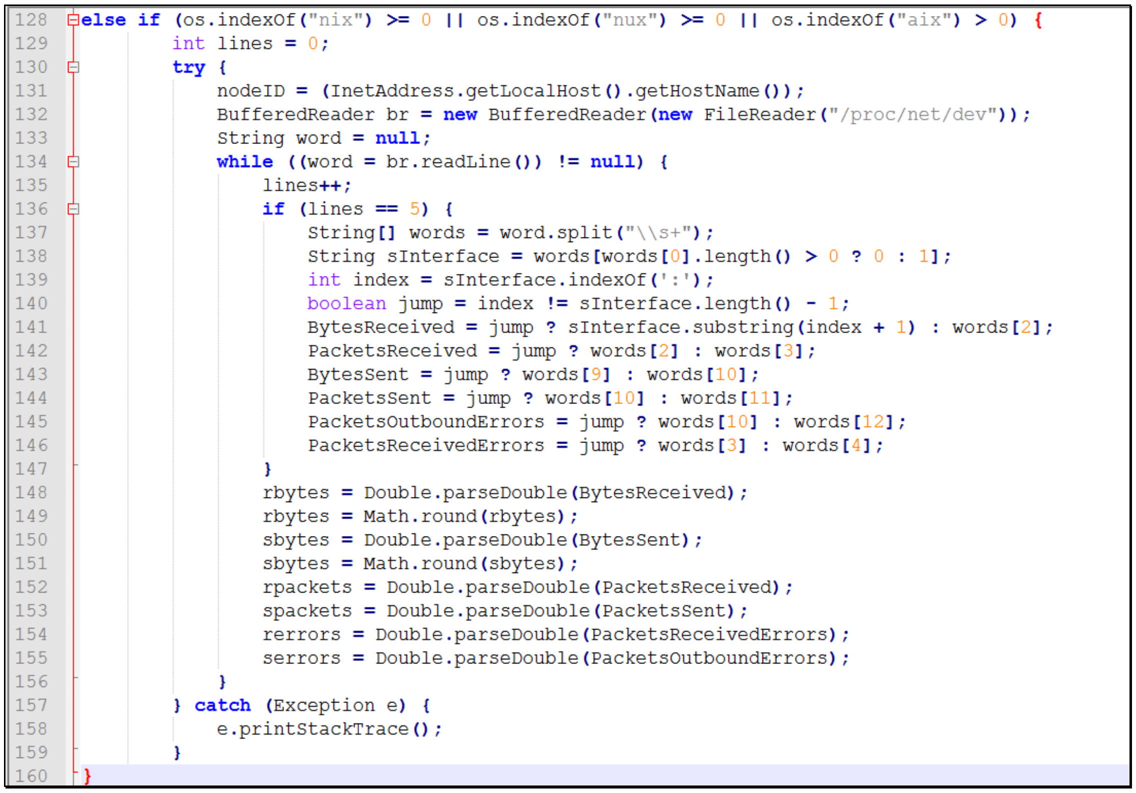
Figure 38: Method That Shows the Process of Killing a Running Process.

Line 214 shows and initializes the class Process to runtime processes of UNIX system files and reads the applications into a string, line 216. The method loops the string line by line to check any unwanted running application, *processName*, and close the application once the *IF* statement becomes true, line 220.

4.3.2. Network Statistics Behaviour

Device network statistics is essential for any NMS to collect so it can be used to manage network resources effectively. This behaviour collects the number of packets sent/received, packet errors and the data consumption downloaded/uploaded and stores it in data storage so it can be associated with the user of a device. The network statistics is associated with the user according to the time the user has been using the device. Because the system is tested on different vendor machines, it first checks the underlying operating system then collects the information according to the different system files. The *NodeAgent's* inference engine uses this data to reason about the network policies. This data is sent to the database of the *MasterAgent* periodically thereby implementing a JADE Cyclic Behaviour.

Figure 39 shows a snippet of a *NodeStats()* method that retrieves network statistics of the managed device. The method first learns the device hostname, line 131, and reads the UNIX file (*proc/net/dev*) that has a network statistics to an input buffer reader, line 132. The code jumps the headers of the text file to the actual values, lines 134, 135 and 135, then manipulates the values and assigns them accordingly.



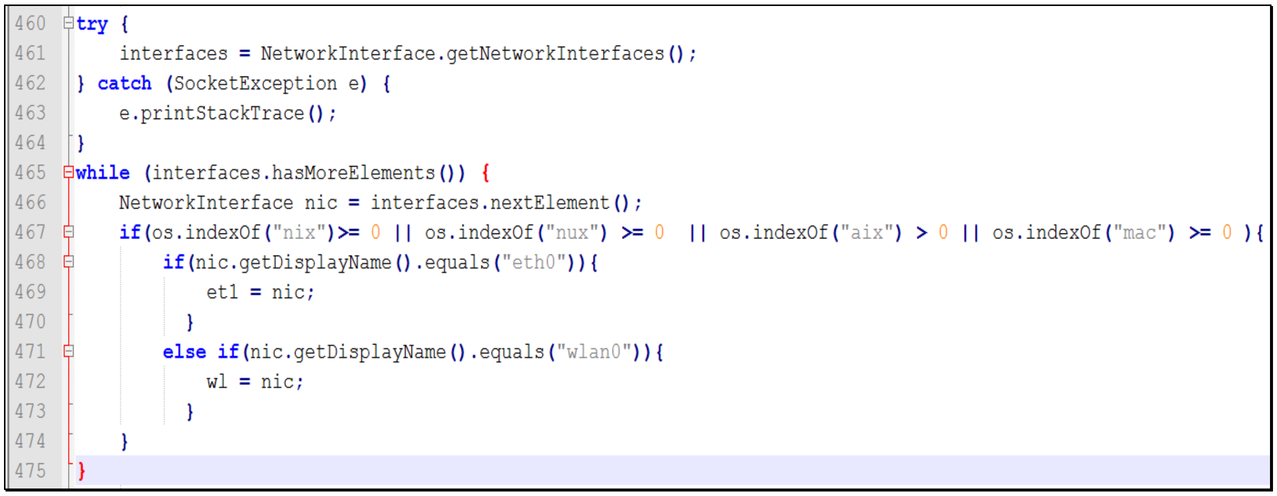
```java
else if (os.indexOf("nix") >= 0 || os.indexOf("nux") >= 0 || os.indexOf("aix") > 0) {
        int lines = 0;
        try {
            nodeID = (InetAddress.getLocalHost().getHostName());
            BufferedReader br = new BufferedReader(new FileReader("/proc/net/dev"));
            String word = null;
            while ((word = br.readLine()) != null) {
                lines++;
                if (lines == 5) {
                    String[] words = word.split("\\s+");
                    String sInterface = words[words[0].length() > 0 ? 0 : 1];
                    int index = sInterface.indexOf(':');
                    boolean jump = index != sInterface.length() - 1;
                    BytesReceived = jump ? sInterface.substring(index + 1) : words[2];
                    PacketsReceived = jump ? words[2] : words[3];
                    BytesSent = jump ? words[9] : words[10];
                    PacketsSent = jump ? words[10] : words[11];
                    PacketsOutboundErrors = jump ? words[10] : words[12];
                    PacketsReceivedErrors = jump ? words[3] : words[4];
                }
                rbytes = Double.parseDouble(BytesReceived);
                rbytes = Math.round(rbytes);
                sbytes = Double.parseDouble(BytesSent);
                sbytes = Math.round(sbytes);
                rpackets = Double.parseDouble(PacketsReceived);
                spackets = Double.parseDouble(PacketsSent);
                rerrors = Double.parseDouble(PacketsReceivedErrors);
                serrors = Double.parseDouble(PacketsOutboundErrors);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
}
```

Figure 39: Method to Show the Collection Network Statistics.

Immediately after this method, the *NodeAgent* sends and updates the values to a local database, and then after a period of time it updates the database in the *MasterAgent*.

4.3.3. Network and Internet Connection Behaviours

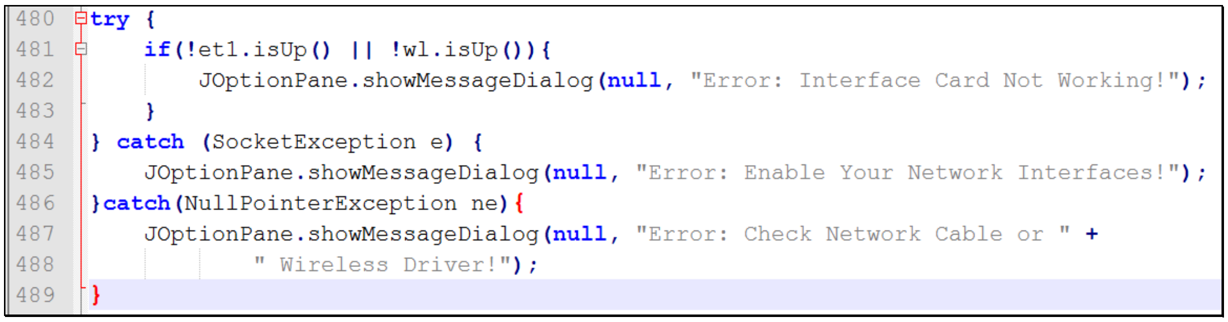
This section discusses how the *NodeAgent* checks if the network device is able to connect to the network and the Internet, it is implemented by first detecting available Network Interface Cards (NICs); testing the connection between the device and the Gateway and then testing the Internet availability. The *NodeAgent* has a *checkInterface()* method that detects if the physical NIC is up or down and a *checkInternet()* method that tests the network and Internet connection. The *checkInterface()* method detects the NIC and if it is not working, it pops up a message on the screen of the user reporting the problem so they can have knowledge about this hardware or network cable problem before the user tries to access the Internet. Figure 40 shows the snippet of the *checkInterface()* method.



```
460 try {
461     interfaces = NetworkInterface.getNetworkInterfaces();
462 } catch (SocketException e) {
463     e.printStackTrace();
464 }
465 while (interfaces.hasMoreElements()) {
466     NetworkInterface nic = interfaces.nextElement();
467     if(os.indexOf("nix")>= 0 || os.indexOf("nux") >= 0  || os.indexOf("aix") > 0 || os.indexOf("mac") >= 0 ){
468         if(nic.getDisplayName().equals("eth0")){
469             et1 = nic;
470           }
471         else if(nic.getDisplayName().equals("wlan0")){
472             wl = nic;
473           }
474     }
475 }
```

Figure 40: Getting Network Interface.

The *checkInterface()* method uses the Java Network API to enumerate the device network interfaces, line 461 and manipulate the interfaces according to the underlying operating system, line 467. The method assigns the names of the interfaces based on either the device uses a Wireless or Ethernet connection, line 468-473. After getting the used interface, the method gathers the information about the interface, Figure 41.

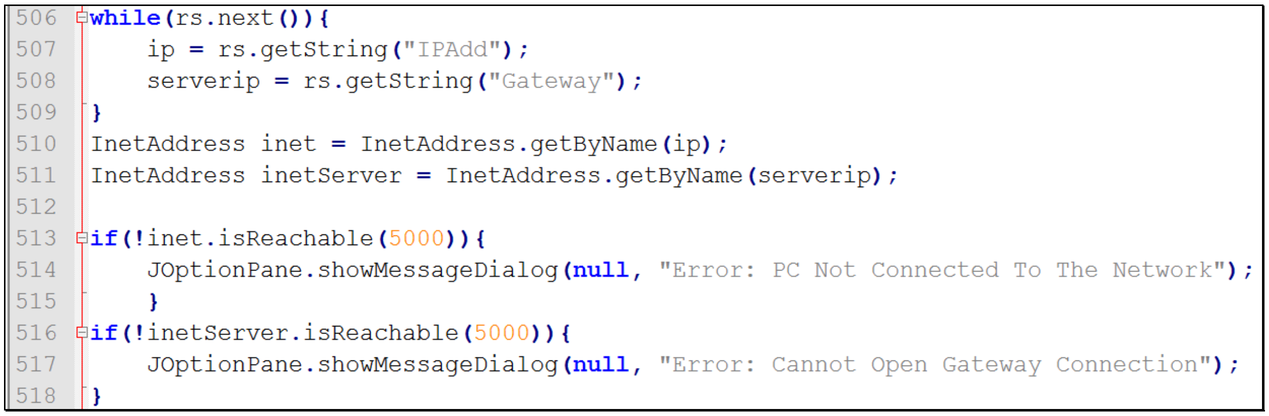


```
480 try {
481     if(!et1.isUp() || !wl.isUp()){
482         JOptionPane.showMessageDialog(null, "Error: Interface Card Not Working!");
483     }
484 } catch (SocketException e) {
485     JOptionPane.showMessageDialog(null, "Error: Enable Your Network Interfaces!");
486 }catch(NullPointerException ne){
487     JOptionPane.showMessageDialog(null, "Error: Check Network Cable or " +
488             " Wireless Driver!");
489 }
```

Figure 41: Testing NIC.

The *checkInterface()* method detects if the wireless or Ethernet NIC is not up with the Java *isUp()* Boolean method, line 481 and displays a message on the screen for the user with the details about the problem, line 482. The method handles the exceptions by troubleshooting the kind of a resulted exception; the socket exception is thrown to indicate that the invoked interface in not active, line 484-486. And the null pointer exception is thrown when the interface is not available at all, line 486-488, this indicates that a user should check if the Ethernet cable is plugged or not or to check if the Wireless connection mode is On/Off.

The network and Internet behaviour has a *checkInternet()* method that tests for the network and Internet connectivity. This method gets the IP and Gateway address to test the network connectivity and uses known site to test Internet connection. This kind of troubleshooting helps users to identify the specific location of a problem. This method retrieves the IP and Gateway address from the device data storage and uses a specific URL (in this case *http://www.google.com)* to test for the Internet connection. There are three options of results from this method, namely: if the *NodeAgent* cannot ping the IP address, that means the device is not connected to local network; if the *NodeAgent* cannot ping the server gateway, that means there is a communication failure between the device and the gateway and lastly if the device cannot connect to the external IP address, that means there is no Internet connection for whole network.

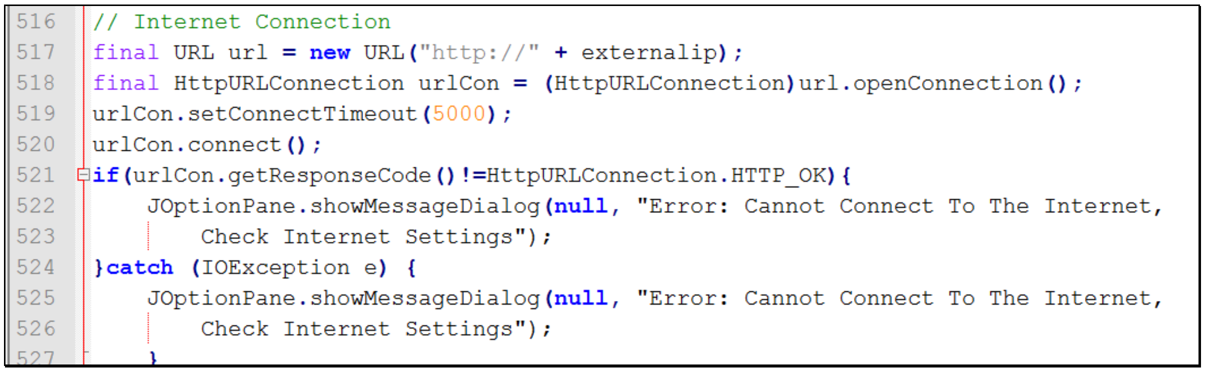


```java
while(rs.next()){
    ip = rs.getString("IPAdd");
    serverip = rs.getString("Gateway");
}
InetAddress inet = InetAddress.getByName(ip);
InetAddress inetServer = InetAddress.getByName(serverip);

if(!inet.isReachable(5000)){
    JOptionPane.showMessageDialog(null, "Error: PC Not Connected To The Network");
    }
if(!inetServer.isReachable(5000)){
    JOptionPane.showMessageDialog(null, "Error: Cannot Open Gateway Connection");
}
```

Figure 42: Ping Local Device and Gateway Implementation.

Figure 42 shows a snippet code of *checkInternet()* method that retrieves the network IP addresses of the device and the gateway, line 506-509 and converts the strings from the database to 32-bit unsigned IP numbers by using Java *InetAddress* class, line 510-511. After resolving the retrieved strings, the method checks if the IP addresses are not reachable by using Java *isReachable()* method and displays a message to the user about the discovered problem, line 513-518. This gives the user a specific indication of connectivity and where the problem is, if the user cannot connect to gateway IP address that explains a link failure between their devices and the gateway.

When the *NodeAgent* does not find any internal network failures and problems, its behaviour periodically checks if the network device is able to connect to the Internet. Figure 43 shows how the agent tests the network connectivity.



```
516  // Internet Connection
517  final URL url = new URL("http://" + externalip);
518  final HttpURLConnection urlCon = (HttpURLConnection)url.openConnection();
519  urlCon.setConnectTimeout(5000);
520  urlCon.connect();
521  if(urlCon.getResponseCode()!=HttpURLConnection.HTTP_OK){
522      JOptionPane.showMessageDialog(null, "Error: Cannot Connect To The Internet,
523          Check Internet Settings");
524  }catch (IOException e) {
525      JOptionPane.showMessageDialog(null, "Error: Cannot Connect To The Internet,
526          Check Internet Settings");
527      }
```

Figure 43: Internet Connectivity Test Implementation.

The *checkInternet()* method takes the URL of the declared site as a test case, line 517 and uses *HttpURLConnection* instance to make a request on Internet connection, line 518. The method sets the connection timeout of five seconds, line 519 and then connects to the URL given, line 520. The method checks if the connection returned false and shows the message to the user about the Internet connectivity failure, line 521-523. The method handles the I/O exception of the given URL by displaying same message of *HttpURLConnection* instance to the user.

4.4. Inference Engine

The system uses Jess rule engine to specify the basic rules, policies and goals of the network, each rule expresses if some statement(s) are true, thereby forming a knowledge base system. A rule engine consists of three parts: facts, rules and actions. The knowledge data sources are the facts captured by the agents; the system administrator specifies rules and actions are executed when the current network error/state meets the rules. This rule engine helps to separate the management code with the business logic of the system and adds the dynamic process of management task since a network is an environment that changes every now and then. This system uses forward chaining method; for it to take actions, it starts with facts/data (current state) then draws conclusions. The following diagram describes this phenomenon, Figure 44.

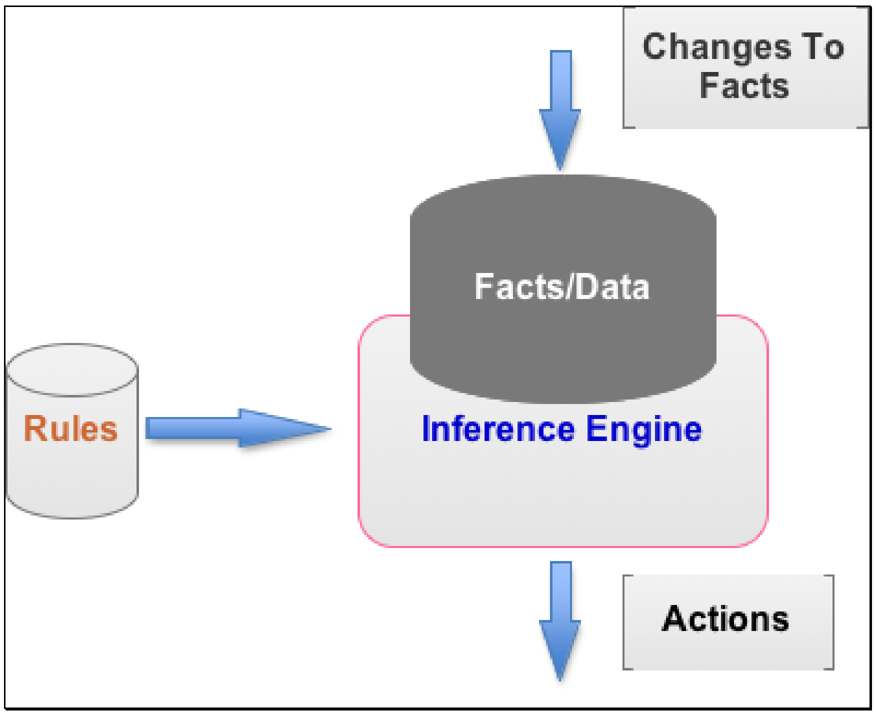
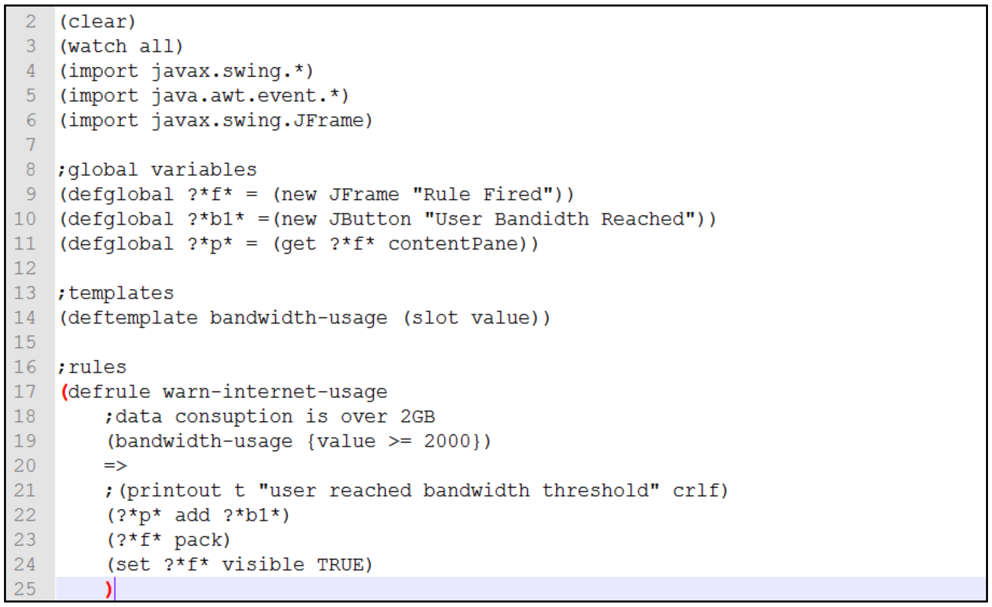


Figure 44: System Inference Engine.

The implementation of our rule engine uses an integration of Java and Jess, Java is used to define facts about the managed device and Jess is used to specify the network rules and execute actions. The rule engine consists of Jess file that contains system rules and actions; and a Java class that collects facts about the current state of the device from a database. The rule engine performs basic check-ups about the device state and executes actions if required to. Jess library has a full access to the Java API and this makes it easy to call Java methods and to write Java code inside a Jess file to perform complicated tasks. The Java class creates an instance of Rete object that allows the manipulation of Jess and then load the rules at runtime. The Rete object has four basic functions: reset the engine back to its initial state, load the device data (facts), execute rules and extract results. The R*uleEngine* class extends JADEs cyclic behavior to meet the four functions of the Rete objects and to ensure that the rule engine runs indefinitely to catch any failures/errors on the device.

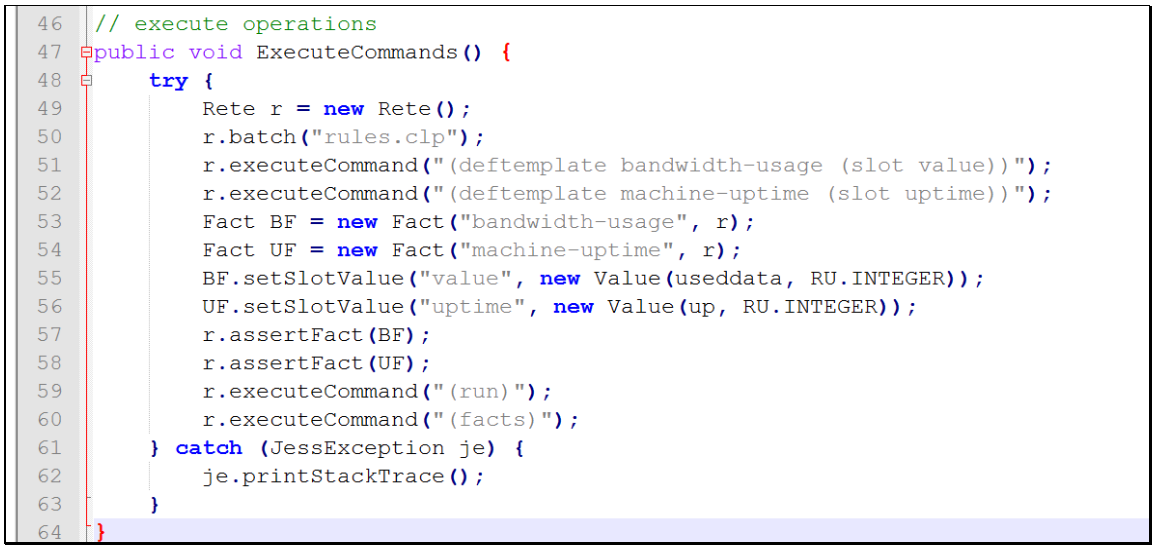
Figure 45 shows an implementation of data consumption monitor rule that checks if the user has reached a two gigabytes threshold and displays a warning on their screen.



```
(clear)
(watch all)
(import javax.swing.*)
(import java.awt.event.*)
(import javax.swing.JFrame)

;global variables
(defglobal ?*f* = (new JFrame "Rule Fired"))
(defglobal ?*b1* =(new JButton "User Bandidth Reached"))
(defglobal ?*p* = (get ?*f* contentPane))

;templates
(deftemplate bandwidth-usage (slot value))

;rules
(defrule warn-internet-usage
    ;data consuption is over 2GB
    (bandwidth-usage {value >= 2000})
    =>
    ;(printout t "user reached bandwidth threshold" crlf)
    (?*p* add ?*b1*)
    (?*f* pack)
    (set ?*f* visible TRUE)
    )
```

Figure 45: Implementation of a Jess file that contains Rules.

The object first initializes the Rete engine, line 2, and then instructs the engine to print all the useful diagnostics of the object with the (*watch all)* method on line 3. This diagnostic information is helpful to check if the LHS conditions of a specific rule have been met by the given list of facts. The object accesses the Java AWT and Swing API and instantiate a JFrame called *"Rule Fired"* with a message dialog that prints a message for the user. Each rule has its specific name called *Template* that is used to associate each rule with its facts and actions. The name of data consumption rule is declared on line 14 (*bandwidth-usage)*, this name is also used by the Java class when inserting the current state of the device to the Rete engine. The LHS of the rule, line 19 Figure 45, checks on the knowledge in hand if the data consumption has reached two gigabytes threshold and if so it displays a message dialog on the user screen telling them that they have reached their data limit. A similar mechanism can also be used to automatically disconnect users from the network when they reached specified usage thresholds.



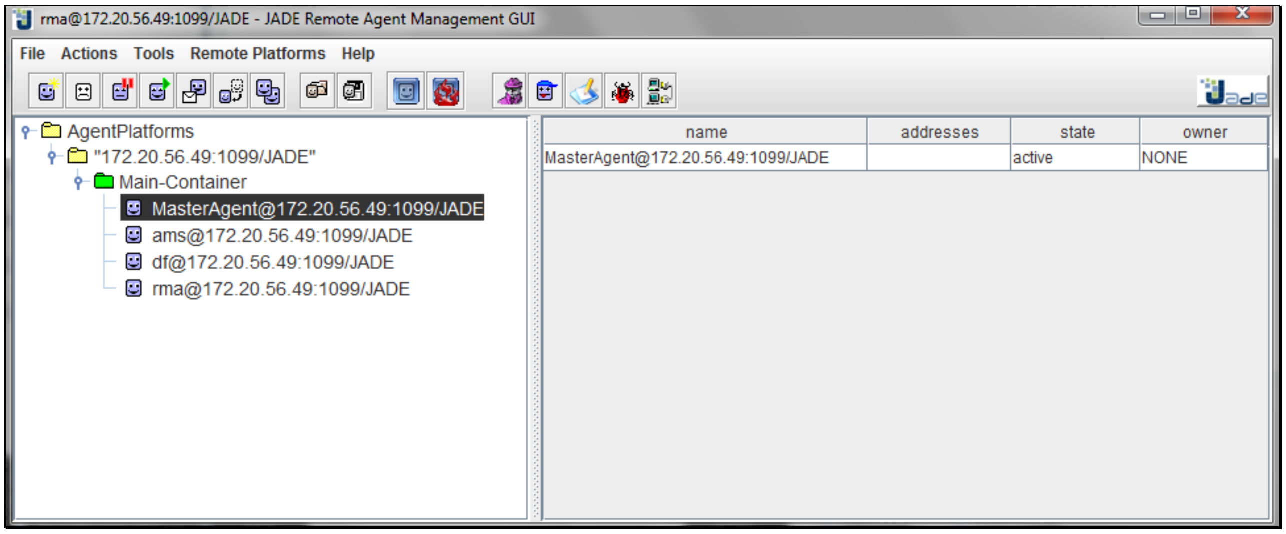
```
46  // execute operations
47  public void ExecuteCommands() {
48      try {
49          Rete r = new Rete();
50          r.batch("rules.clp");
51          r.executeCommand("(deftemplate bandwidth-usage (slot value))");
52          r.executeCommand("(deftemplate machine-uptime (slot uptime))");
53          Fact BF = new Fact("bandwidth-usage", r);
54          Fact UF = new Fact("machine-uptime", r);
55          BF.setSlotValue("value", new Value(useddata, RU.INTEGER));
56          UF.setSlotValue("uptime", new Value(up, RU.INTEGER));
57          r.assertFact(BF);
58          r.assertFact(UF);
59          r.executeCommand("(run)");
60          r.executeCommand("(facts)");
61      } catch (JessException je) {
62          je.printStackTrace();
63      }
64  }
```

Figure 46: Method that Asserts facts to Rete Object.

Figure 46 shows the Java class that loads a file with rules (*rules.clp*), keeps on retrieving the facts from a database and then asserts facts to the Rete object. The *ExecuteCommands()* method loads Jess file on a running instance of Rete engine, line 50, and declares a *bandwidth-usage* template, line 51, that has the same name as in Jess file with rules. The method on line 55 inserts the retrieved value (fact) on the *bandwidth-usage* template and asserts the fact on the Rete engine, line 57. Since the *RuleEngine* class extends the JADEs cyclic behavior, it periodically accesses the database to track any device changes thereby providing resource management function and allowing the Rete engine to perform its four main functions.

4.5. Starting the Agent Platform

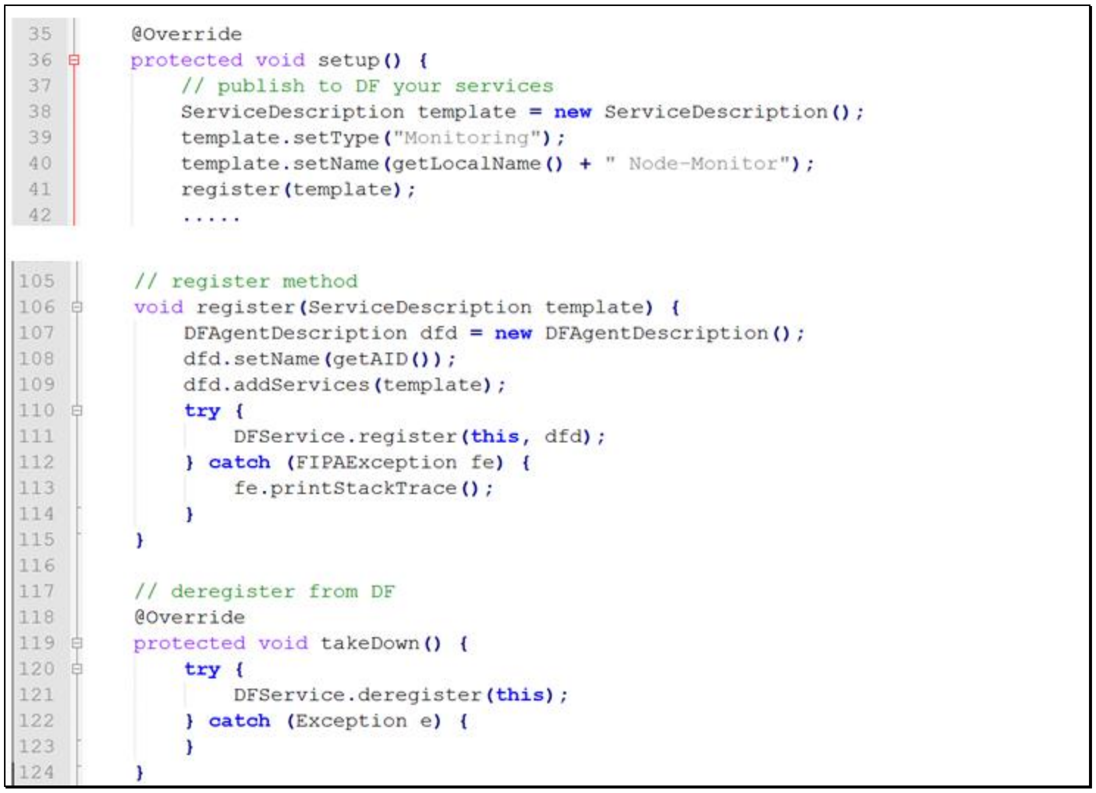
The agent platform provides the runtime environment for agents to operate on the network devices and this requires Java and JADE platform enabled devices. The agent platform contains and runs the main-container that has the AMS, *MasterAgent*, DF and the Remote Monitoring Agent to create the agent platform. The agent platform resides in a network device and consists of containers to form a distributed system. These containers consist of running agents and each container resides on a separate network device thereby forming a MAS. The *MasterAgent* has a GUI that lists all the operating containers and IP addresses of devices they are operating on. Each agent operating on the agent platform has its own specific AID. When the agent platform starts, it starts the *MasterAgent* and registers the services it offers to the agent platform DF. Then every agent (*NodeAgent*) that is started on each network devices is added to one running instance of an agent platform and also registers with the agent platform's DF. Figure 47 shows instantiation of the *MasterAgent* class that extends *jade.core.Agent* thereby starting the agent platform GUI that resides on host 172.20.56.49; every agent (*NodeAgent*) that is started on the agent platform will use this IP address as a host of a main-container. The agent platform also comes with useful agents for example Sniffer, Dummy and DF agent. Sniffer agent is the agent in the agent platform that shows the main tasks of the *MasterAgent* class in its GUI. DummyAgent is used to send, receive and inspect ACL messages from/to agents and save/read messages from/to file. The DF agent provides a centralized registry of agents associated with their service description. The DF GUI allows a user to register, deregister, modify and search agent descriptions in the agent platform. The DF also provides "yellow pages" services to all agents in the agent platform and only one instance of a DF can exist in an agent platform and a DF can register with other DF's on different agent platforms to form a federation of DF's thereby providing a redundant system.


rma@172.20.56.49:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help
AgentPlatforms
"172.20.56.49:1099/JADE"
Main-Container
MasterAgent@172.20.56.49:1099/JADE
ams@172.20.56.49:1099/JADE
df@172.20.56.49:1099/JADE
rma@172.20.56.49:1099/JADE

name	addresses	state	owner
MasterAgent@172.20.56.49:1099/JADE		active	NONE

Figure 47: Starting the Agent Platform.

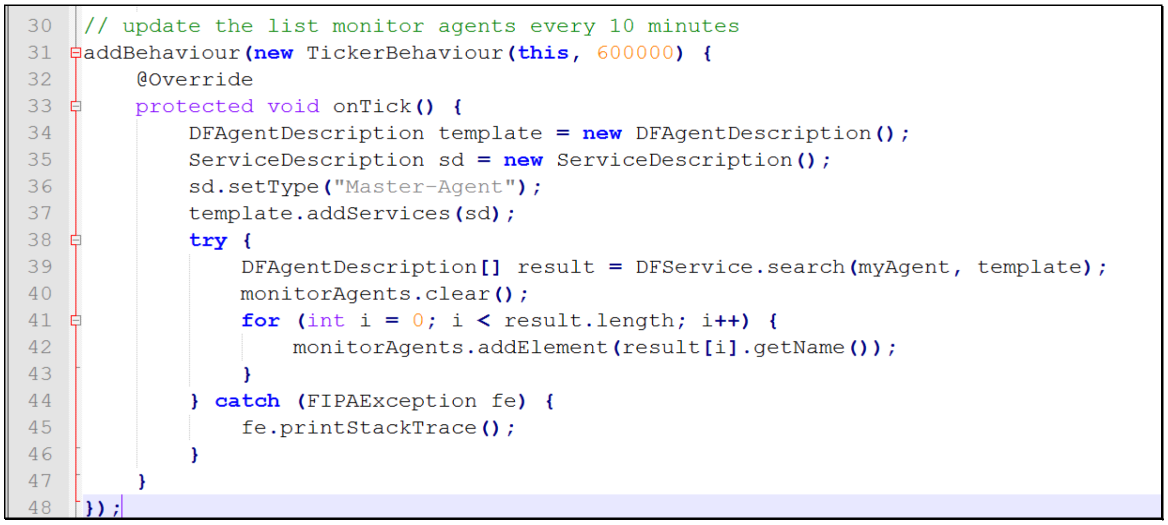
Figure 48 shows an implementation of agent registration with the DF, the first snippet (line 35-41) of the code in the setup method creates an agent service description *Monitoring* and registers this service using the *register()* method. On Figure 48, line 106-115 shows the *register()* method that declares a DF agent description and adds the agent identifier to the agent service description. Line 119-124 overrides an *jade.core.Agent* class *takedown()* method that removes the agent services in the DF when the agent dies/stop. An agent only performs the task of registering its service to the DF once when it is started. The *MasterAgent* has a duty to refresh the DF every 10 minutes in case a new agent has been started or died. This agent implements JADEs Ticker behaviour to facilitate the refreshment of the agent's DF.



```java
@Override
protected void setup() {
    // publish to DF your services
    ServiceDescription template = new ServiceDescription();
    template.setType("Monitoring");
    template.setName(getLocalName() + " Node-Monitor");
    register(template);
    .....

// register method
void register(ServiceDescription template) {
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(getAID());
    dfd.addServices(template);
    try {
        DFService.register(this, dfd);
    } catch (FIPAException fe) {
        fe.printStackTrace();
    }
}

// deregister from DF
@Override
protected void takeDown() {
    try {
        DFService.deregister(this);
    } catch (Exception e) {
    }
}
```

Figure 48: Method to Show Registration of an Agent to DF.

Figure 49 shows the *MasterAgent* ticker behaviour method that refreshes the list of *NodeAgents* in the DF. Line 31 declares a ticker behaviour that refreshes at 600000 milliseconds and at line 31 the method searches the service in the DF using the *MasterAgent* template *"Master-Agent"*. And then it inserts all the *NodeAgent* services found in the DF to a vector array *monitorAgents*, line 38-42.



```
// update the list monitor agents every 10 minutes
addBehaviour(new TickerBehaviour(this, 600000) {
    @Override
    protected void onTick() {
        DFAgentDescription template = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType("Master-Agent");
        template.addServices(sd);
        try {
            DFAgentDescription[] result = DFService.search(myAgent, template);
            monitorAgents.clear();
            for (int i = 0; i < result.length; i++) {
                monitorAgents.addElement(result[i].getName());
            }
        } catch (FIPAException fe) {
            fe.printStackTrace();
        }
    }
});
```

Figure 49: Behaviour to Refresh the Agent DF.

Once the agent platform is started it is ready to accept remote containers to be added on it (forming a distributed system) and to send/receive messages from other agent platforms or agents. The following subsection describes the implementation of communication method between *MasterAgent* and *NodeAgents* in the agent platform.

4.5.1. Agent Communication Language

For agents to communicate something that makes sense, there is a need to create a certain degree of commonality in terms of communication language, vocabulary and protocols. The system defines an ontology, *NetworkOntology*, for the system agents to exchange messages using a standard FIPA format. The FIPA ACL message format is characterized as follows:

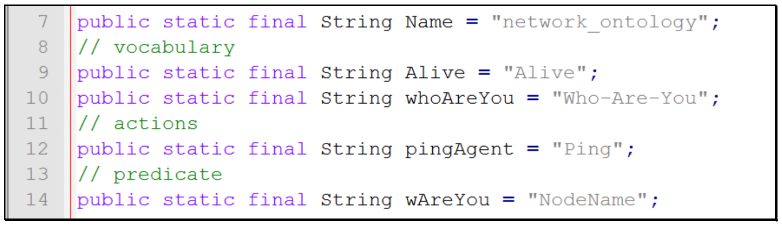
- Intention: REQUEST, INFORM and QUERY_REF etc.
- Attendees: Sender and set of receivers.
- Content: Information exchanged.

- Content Description: Description of (i) the content language to express the content and (ii) the ontology by means of which the agents ascribe a proper meaning to the terms used in the content.

- Conversation control: Interaction protocol and conversation identification.

The implementation of agent ontology also allows JADE agents to interoperate with other agent systems. The information exchanged between our system agents is encoded with the help of the SL and decoded upon arrival by the intended receiver(s). FIPA Semantic Language (SL) language is a human readable string-encoded content language used by software agents.

In this model, communication happens through the exchange of asynchronous messages between agents corresponding to communicative acts, for example SUSCRIBE, INFORM, PROPOSE, and REQUEST_WHEN etc. The following shows an implementation of an ACL message sent by the one *NodeAgent* to all the agents in the agent platform to inform them about its existence: The system first defines an application-specific ontology NetworkOntology that extends *BasicOntology* and implements *ConceptSchema*, *AgentActionSchema* and *PredicateSchema* interfaces provided by JADE. These three interfaces describe the structure of concepts, actions and predicates that are allowed for an agent to create a message.

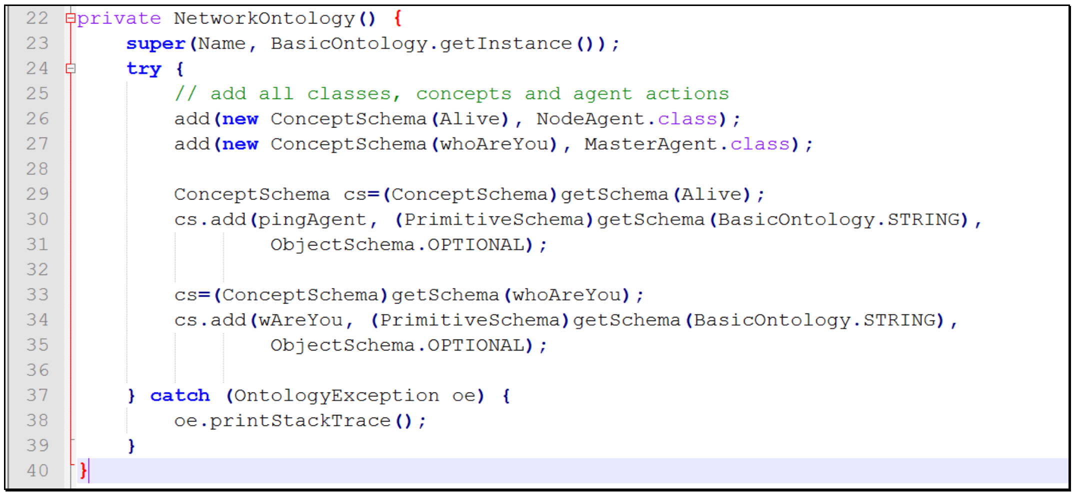


```
public static final String Name = "network_ontology";
// vocabulary
public static final String Alive = "Alive";
public static final String whoAreYou = "Who-Are-You";
// actions
public static final String pingAgent = "Ping";
// predicate
public static final String wAreYou = "NodeName";
```

Figure 50: Definition of ACL Message Structure.

Figure 50 shows the instantiation of the ontology name and the structure of the message that consists of agent vocabulary, action and predicate. Then the ontology adds the vocabulary to the

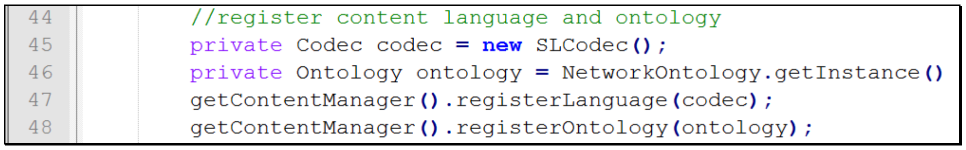
MasterAgent and *NodeAgent* class, line 26-27 Figure 51. On line 29, the ontology inserts the *Alive* item to a *ConceptSchema* and adds an action to it, line 30-31.



```
22 private NetworkOntology() {
23     super(Name, BasicOntology.getInstance());
24     try {
25         // add all classes, concepts and agent actions
26         add(new ConceptSchema(Alive), NodeAgent.class);
27         add(new ConceptSchema(whoAreYou), MasterAgent.class);
28 
29         ConceptSchema cs=(ConceptSchema)getSchema(Alive);
30         cs.add(pingAgent, (PrimitiveSchema)getSchema(BasicOntology.STRING),
31                 ObjectSchema.OPTIONAL);
32 
33         cs=(ConceptSchema)getSchema(whoAreYou);
34         cs.add(wAreYou, (PrimitiveSchema)getSchema(BasicOntology.STRING),
35                 ObjectSchema.OPTIONAL);
36 
37     } catch (OntologyException oe) {
38         oe.printStackTrace();
39     }
40 }
```

Figure 51: Structure of the *NetworkOntology*.

Then the *NodeAgent* registers the defined ontology and language to its content manager before it can create the message, line 47-48 Figure 52. To encode and decode messages, system agents use the JADE SL codec.

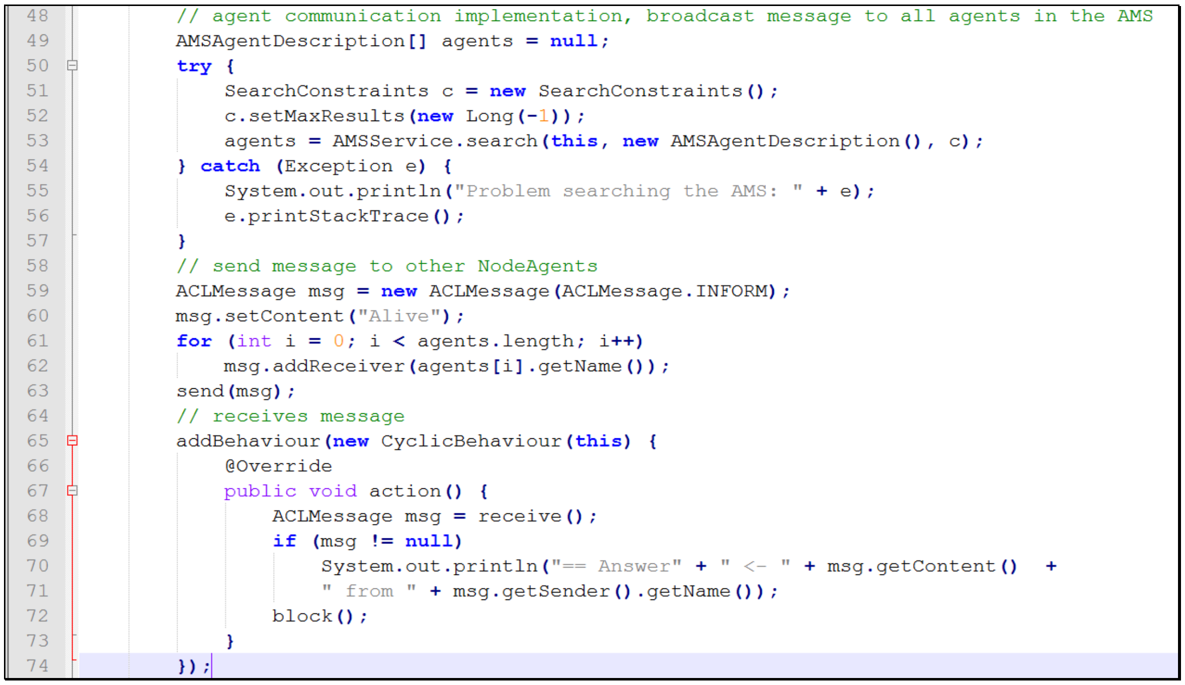


```
44         //register content language and ontology
45         private Codec codec = new SLCodec();
46         private Ontology ontology = NetworkOntology.getInstance()
47         getContentManager().registerLanguage(codec);
48         getContentManager().registerOntology(ontology);
```

Figure 52: Language and Ontology Registration.

The *NodeAgent* informs the agents with the message "*Alive*" by first searching the AMS for agents that are in the agent platform. On Figure 53 line 49 declares an array of AMS agent description and models a search constraint that has negative value to add all the agents in the agent platform, line 51-53. The agent composes an inform message and adds all the receivers to the AMS agent description array then sends the messages using the JADE *SEND* method, line

59-63.



```
48      // agent communication implementation, broadcast message to all agents in the AMS
49      AMSAgentDescription[] agents = null;
50      try {
51          SearchConstraints c = new SearchConstraints();
52          c.setMaxResults(new Long(-1));
53          agents = AMSService.search(this, new AMSAgentDescription(), c);
54      } catch (Exception e) {
55          System.out.println("Problem searching the AMS: " + e);
56          e.printStackTrace();
57      }
58      // send message to other NodeAgents
59      ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
60      msg.setContent("Alive");
61      for (int i = 0; i < agents.length; i++)
62          msg.addReceiver(agents[i].getName());
63      send(msg);
64      // receives message
65      addBehaviour(new CyclicBehaviour(this) {
66          @Override
67          public void action() {
68              ACLMessage msg = receive();
69              if (msg != null)
70                  System.out.println("== Answer" + " <- " + msg.getContent()  +
71                  " from " + msg.getSender().getName());
72              block();
73          }
74      });
```

Figure 53: Agent Communication Implementation.

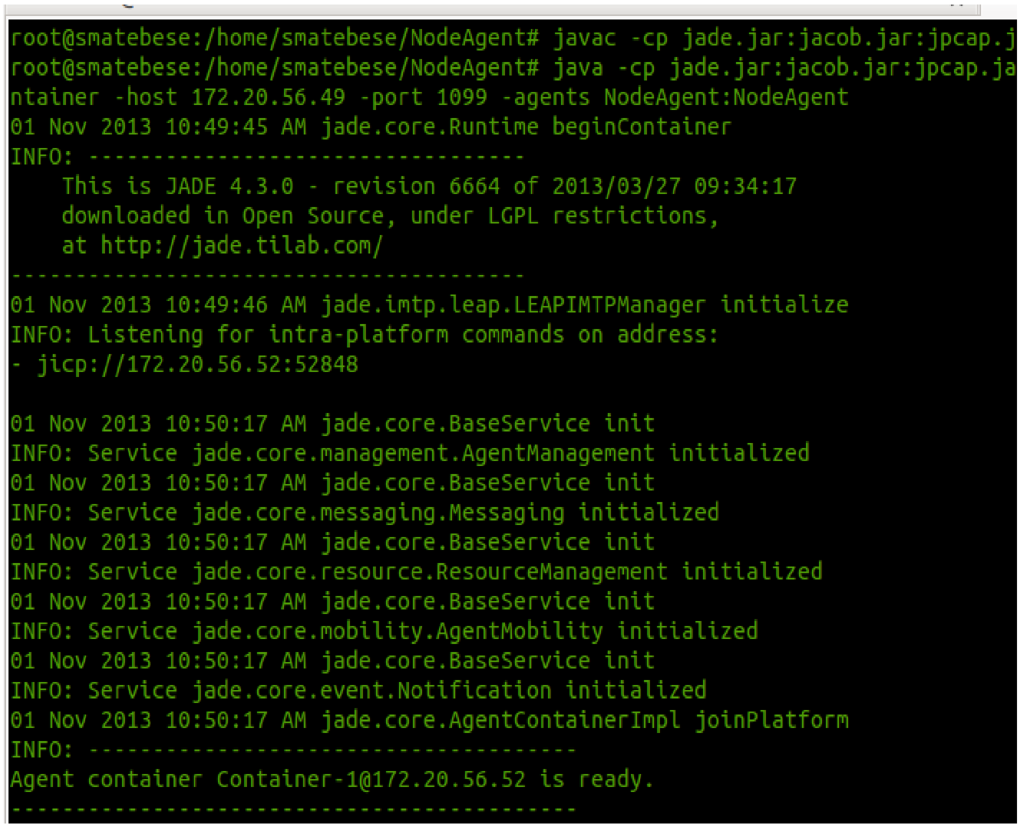
The agent implements a cyclic behaviour method to prepare it to handle the incoming messages anytime any other agent sends it a message. The method receives the content of the message and the message sender name, line 70-71. The *block()* method tells an agent to only call the *action()* method again if a message is received. These messages are transported between agents in the agent platform or in different platform using FIPA MTP that is provided by an ACC. An ACC is responsible for sending and receiving messages in the agent platform according to the transport instructions contained in the message envelope. The message envelope has ten parameters and the ACC can only read these parameters but not the message content. Table 3 describes the message envelope parameters [43].

Table 3: Message Envelope Parameters.

Envelope Parameter	**Description**
To	Message recipient(s) name
From	Message sender name
Comments	Message comments
ACL-Representation	Syntax representation of the message payload
Payload-length	Number of bytes for the message payload
Payload-encoding	Language encoding of the message payload
Date	Message envelope creation date and time
Intended-receiver	AID of the agent(s) to whom the message is to be delivered
Received	Stamp to indicate the receipt of a message by an ACC
Transport-behaviour	Transport requirements of the message

4.5.2. Creation of a Distributed System

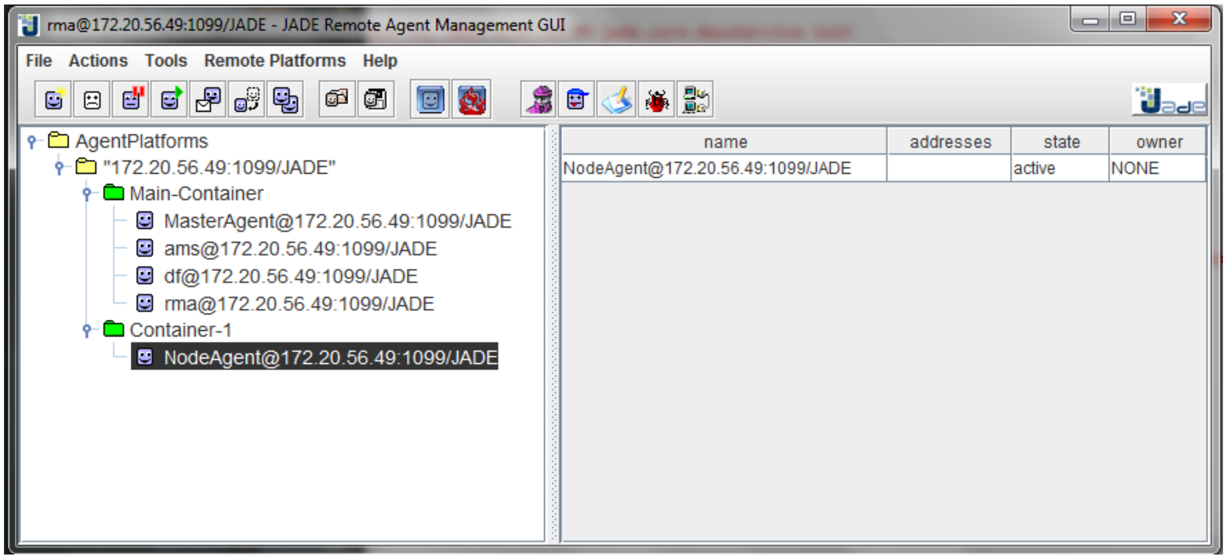
This section shows how system agents are started and then added to the agent platform to form MAS. The *NodeAgents* are added to the agent platform from the network devices thereby forming a fully distributed system. The *NodeAgent* is started on the managed device's terminal; we first compile all the necessary classes including the technologies that are required, Figure 54.



```
root@smatebese:/home/smatebese/NodeAgent# javac -cp jade.jar:jacob.jar:jpcap.j
root@smatebese:/home/smatebese/NodeAgent# java -cp jade.jar:jacob.jar:jpcap.ja
ntainer -host 172.20.56.49 -port 1099 -agents NodeAgent:NodeAgent
01 Nov 2013 10:49:45 AM jade.core.Runtime beginContainer
INFO: ----------------------------------
    This is JADE 4.3.0 - revision 6664 of 2013/03/27 09:34:17
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.tilab.com/
----------------------------------------
01 Nov 2013 10:49:46 AM jade.imtp.leap.LEAPIMTPManager initialize
INFO: Listening for intra-platform commands on address:
- jicp://172.20.56.52:52848

01 Nov 2013 10:50:17 AM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
01 Nov 2013 10:50:17 AM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
01 Nov 2013 10:50:17 AM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialized
01 Nov 2013 10:50:17 AM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
01 Nov 2013 10:50:17 AM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
01 Nov 2013 10:50:17 AM jade.core.AgentContainerImpl joinPlatform
INFO: --------------------------------------
Agent container Container-1@172.20.56.52 is ready.
--------------------------------------------
```

Figure 54: Starting the *NodeAgent* on Managed Device Terminal.

Figure 54 shows how the *NodeAgent* is started on host *172.20.56*.52 and added to agent platform on host *172.20.56.49*. On the host that is running the agent platform with AMS GUI, a new container *Container-1* is added with the agent that is on active state, Figure 55.With the implementation of the agent communication (show in the previous section), these agents can now send each other messages directly using the common language defined.


rma@172.20.56.49:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help

AgentPlatforms
"172.20.56.49:1099/JADE"
Main-Container
MasterAgent@172.20.56.49:1099/JADE
ams@172.20.56.49:1099/JADE
df@172.20.56.49:1099/JADE
rma@172.20.56.49:1099/JADE
Container-1
NodeAgent@172.20.56.49:1099/JADE

name	addresses	state	owner
NodeAgent@172.20.56.49:1099/JADE		active	NONE

Figure 55: Creation of a Distributed System.

To add more agents and devices to the agent platform, one follows the operation presented in Figure 54 and performs the commands on every network device.

4.6. Conclusion

This chapter followed the design method on Chapter 3 to implement a distributed network management system. The implementation process was explained with the use of some extract codes to highlight the major components of a network management system. This chapter showed how the technologies were used and integrated to form decentralized and flexible MAS. The following chapter shows the process of system testing by using test case and experiments thereby evaluating the system.

5. Testing and Evaluation

This chapter presents an end-to-end testing of functional requirements of this system. This section considers the functional, usability and performance testing of our system. This chapter presents test cases that relate back to functional requirement components discussed in Chapter 3. The main idea of this chapter is to test the code integration of the whole system from the MasterAgent to the *NodeAgent.*

5.1. Introduction

The purpose of this chapter is to check if the system conforms to the functional requirements specified. The system test will specifically consider performance management, agent communication, rule engine, data storage and analysis, and the overall system functionality. The performance test plan checks for data consumption and user system uptime threshold values; network interface detection, file system detection, CPU times and alerts when threshold are exceeded. The agent communication test plan will test if the agents in the agent platform are able to send/receive messages using the language and the ontology defined. The rule engine is tested to check if it is able to execute the given rules when the facts about the current state are true. What makes network management possible is a good monitoring tool; therefore the data storage must be able to collect and store the necessary information. This data storage must be accessible from all the managed devices in the network so they can be able to update the database. The system functionality test will check if the *MasterAgent* and *NodeAgent* are able to work together to achieve the network management goal.

5.2. Performance Management Test

This section evaluates the management of network resources and system hardware of the managed devices. First we test if the system is able to monitor the system bandwidth of a selected interface from the *Bandwidth Meter* GUI. The system is supposed, when the network interface is selected, to calculate the consumed data in Kilobytes over time in seconds of that interface and dynamically show it in the GUI. The bandwidth meter measures the achieved throughput, which is the average rate of successful data handover over a communication path [72]. Figure 56 shows the *Bandwidth Meter* GUI that runs on the *MasterAgent*, a user chose to monitor the Ethernet network interface option and clicked the *Start* button on the GUI. Then the GUI starts capturing network packets information on that interface using a multi-platform library Hyperic-Sigar.

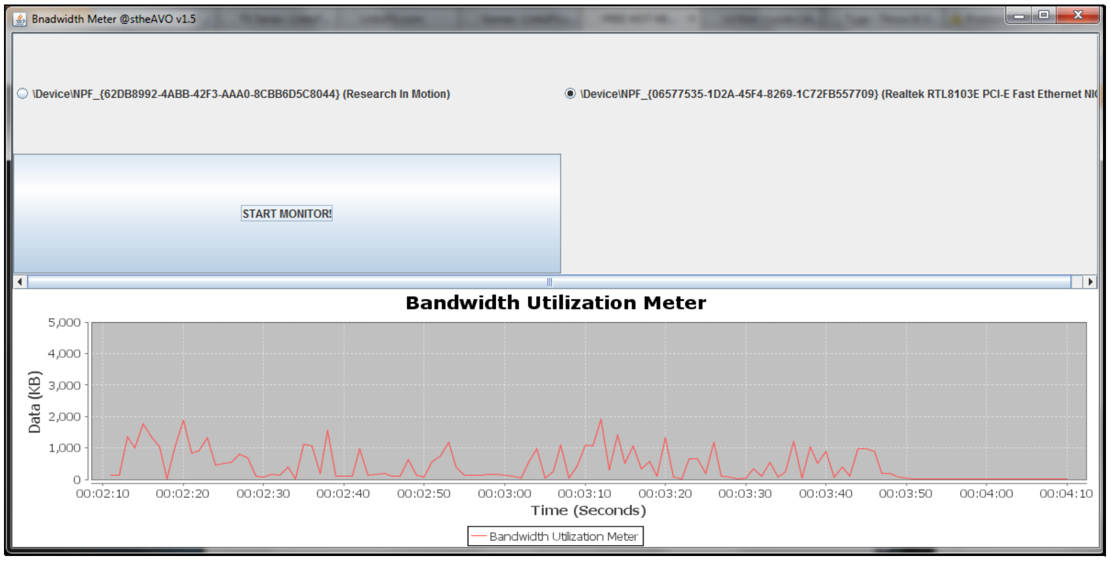

Bnadwidth Meter @stheAVO v1.5
\Device\NPF_{62DB8992-4ABB-42F3-AAA0-8CBB6D5C8044} (Research In Motion)
\Device\NPF_{06577535-1D2A-45F4-8269-1C72FB557709} (Realtek RTL8103E PCI-E Fast Ethernet NI
START MONITOR!
Bandwidth Utilization Meter
Data (KB)
5,000
4,000
3,000
2,000
1,000
0
00:02:10
00:02:20
00:02:30
00:02:40
00:02:50
00:03:00
00:03:10
00:03:20
00:03:30
00:03:40
00:03:50
00:04:00
00:04:10
Time (Seconds)
Bandwidth Utilization Meter

Figure 56: Test for a Bandwidth Meter.

Figure 56 shows that the operation of real-time bandwidth monitoring is possible and successful. There is a need to check the hardware state of the managed devices. This part test, if the system is able to detect the opened system files, retrieves CPU times, shows network interfaces and system uptime. The first phase of monitoring the system health is to retrieve the information of interest from the managed device and the second phase is to analyze and present it to an administrator. The information of interest is stored to database tables Figure 31 and Figure 36; and then an administrator checks the managed device hardware and user information in a web-interface and views working network interface statistics in a GUI. The command to retrieve the hardware and user information is executed inside a Java servlet connecting to a JDBC database when an administrator chooses which device they want to monitor. The web interface refreshes the information every 30 seconds and Figure 32 shows an administrator viewing the device information of user *stheAVO*. Figure 32 shows the system uptime, and lists the entire available network interfaces on the device. It also shows that the user currently uses the Ethernet cable to connect to the Internet. Figure 32 also shows the folders that are opened by the user, CPU times and the device memory with RAM size. The system keeps the inventory of device hardware information; it collects the managed device hardware information and stores it in the database as indicated on Figure 32. The web-interface also shows the device network information that includes the device MAC and IP address, network mask, gateway and Domain Name System (DNS) server address.

The system has a responsibility to archive data consumption for every day of the week for data usage comparison. Figure 57 and Figure 58 shows the data consumed (sent/received in MB) and network packets (sent/received) over weekdays.

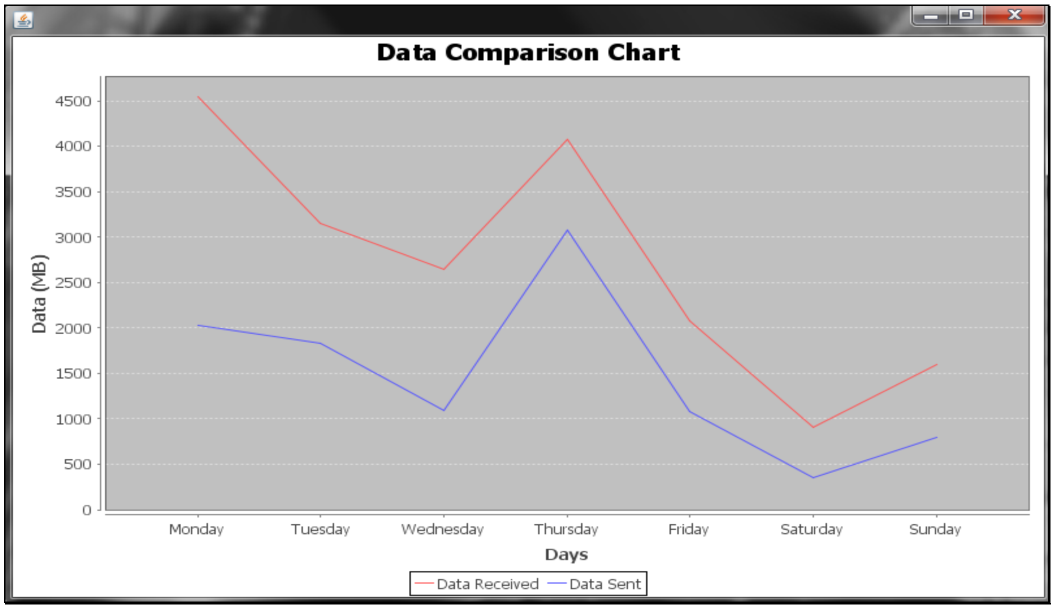


Figure 57: Data Consumed (data in MB) in Days Comparisons.

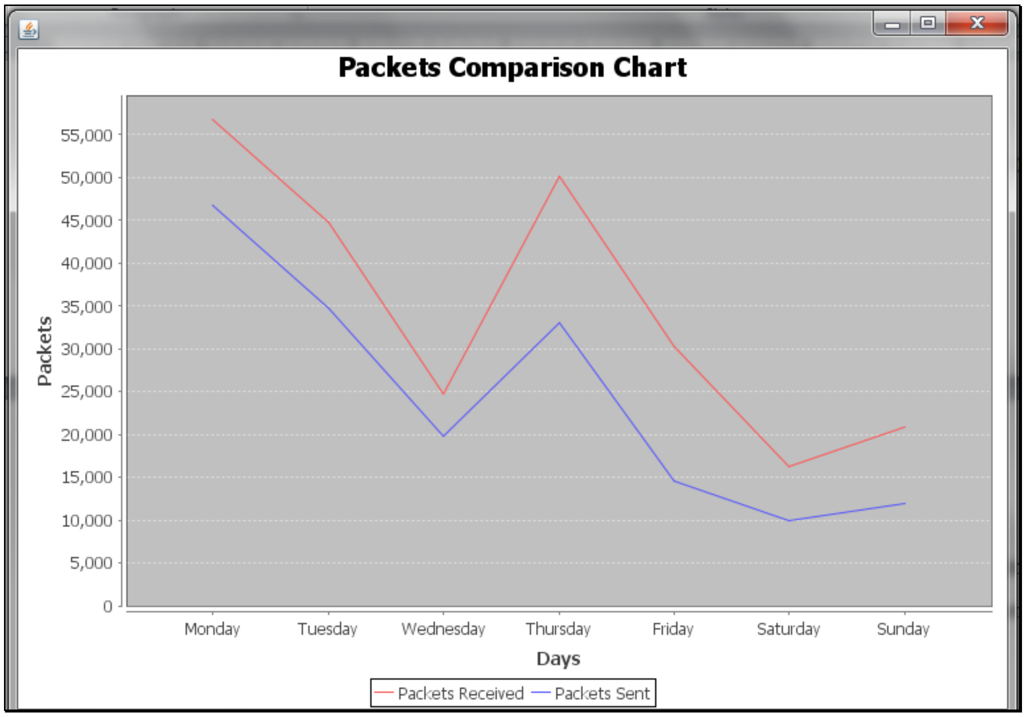


Figure 58: Data Consumed (Packets in numbers) in Days Comparisons.

The reason for data archiving is to check how much data the users consume over a period of time so we can optimize the network performance as time goes by and to check on which days users require most utilization of network resources.

5.3. Agent Communication Test

This section reviews the communication among the agents in the system; agents inform each other about their existence, ping each other to check if they are still alive or requesting agent information and services. The ontology and SL codec defined on the agent platform helps agents to ‘speak’ a common language. This communication test will show an exchange of messages among the *MasterAgent*, *NodeAgent* and the AMS through the *SnifferAgent*. The *MasterAgent* searches for all *NodeAgents* in the agent platform, adds them as recipients, composes a message that queries for their actual names and username of the device they are operating on and sends it.

Figure 59 shows success of message exchange in *SnifferAgent* between the AMS, *MasterAgent*, and the *NodeAgent*.

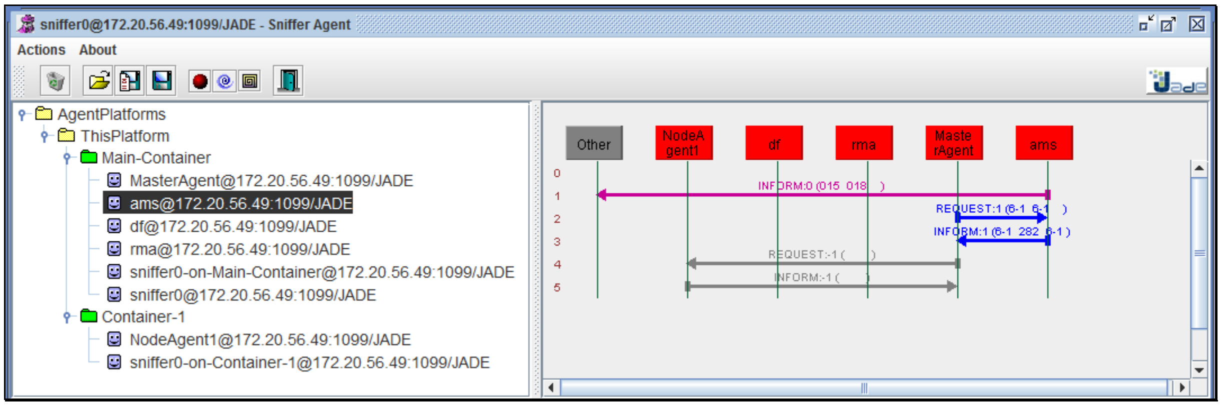
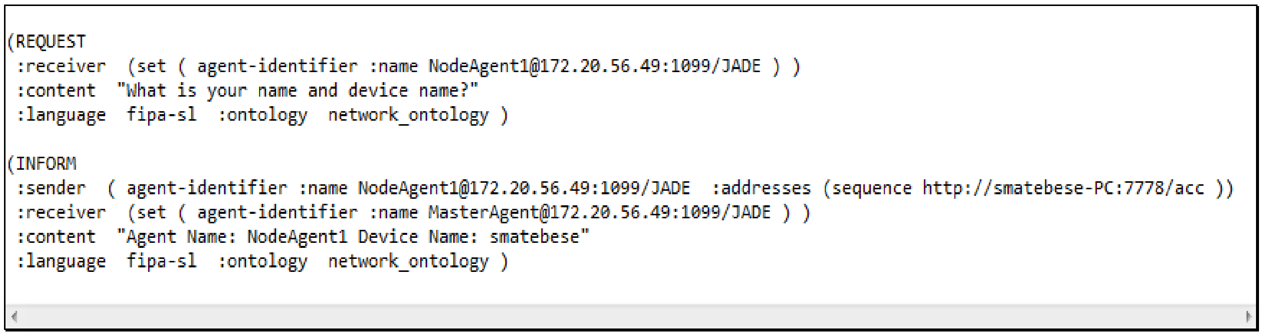

sniffer0@172.20.56.49:1099/JADE - Sniffer Agent
Actions
About
AgentPlatforms
ThisPlatform
Main-Container
MasterAgent@172.20.56.49:1099/JADE
ams@172.20.56.49:1099/JADE
df@172.20.56.49:1099/JADE
rma@172.20.56.49:1099/JADE
sniffer0-on-Main-Container@172.20.56.49:1099/JADE
sniffer0@172.20.56.49:1099/JADE
Container-1
NodeAgent1@172.20.56.49:1099/JADE
sniffer0-on-Container-1@172.20.56.49:1099/JADE
Other
NodeAgent1
df
rma
MasterAgent
ams
INFORM:0
REQUEST:1
INFORM:1
REQUEST:-1
INFORM:-1

Figure 59: *SnifferAgent* Shows Message Exchange in the Agent Platform.

The *SnifferAgent* allows the administrator to track exchanged messages in the agent platform using a structure similar to *Unified Modeling Language (UML) Sequence Diagrams*. On Figure 59 the MasterAgent sends a *REQUEST* message to the AMS that requests for a list of all the *NodeAgents* registered to the platform. The AMS replies with the list requested. The *MasterAgent* takes all the agents given by the AMS and adds them as message receivers, composes a request message for the recipients and then sends it. The *NodeAgent* receives a message from the *MasterAgent* and processes it appropriately and then responds with the required information. The *NodeAgent* implements a cyclic behaviour that waits for incoming messages and processes them accordingly. Figure 60 describes the message contents with the recipients, language and ontology used. The *MasterAgent* sends a *REQUEST* type of communicative act and the *NodeAgent* replies with the *INFORM* message type.



```
(REQUEST
 :receiver  (set ( agent-identifier :name NodeAgent1@172.20.56.49:1099/JADE ) )
 :content  "What is your name and device name?"
 :language  fipa-sl  :ontology  network_ontology )

(INFORM
 :sender  ( agent-identifier :name NodeAgent1@172.20.56.49:1099/JADE  :addresses (sequence http://smatebese-PC:7778/acc ))
 :receiver  (set ( agent-identifier :name MasterAgent@172.20.56.49:1099/JADE ) )
 :content  "Agent Name: NodeAgent1 Device Name: smatebese"
 :language  fipa-sl  :ontology  network_ontology )
```

Figure 60: Message Contents between Agents.

The language and ontology defined eases the handling of ACL message contents between agents and helps system agents to perform content semantic checks thereby providing support for meaningful conversations. Basically the ontology authenticates the information to be converted from the semantic point of view and the language codecs translates the ACL message strings into Java objects. Figure 61 describes this phenomenon, the message is easy to transfer as strings or sequence of bytes and easy for the agent to manipulate as Java objects.

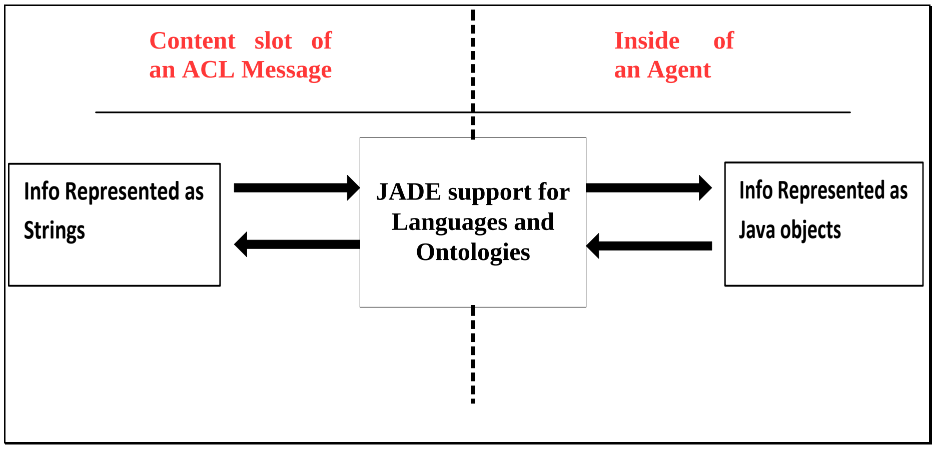

Content slot of an ACL Message
Inside of an Agent
Info Represented as Strings
JADE support for Languages and Ontologies
Info Represented as Java objects

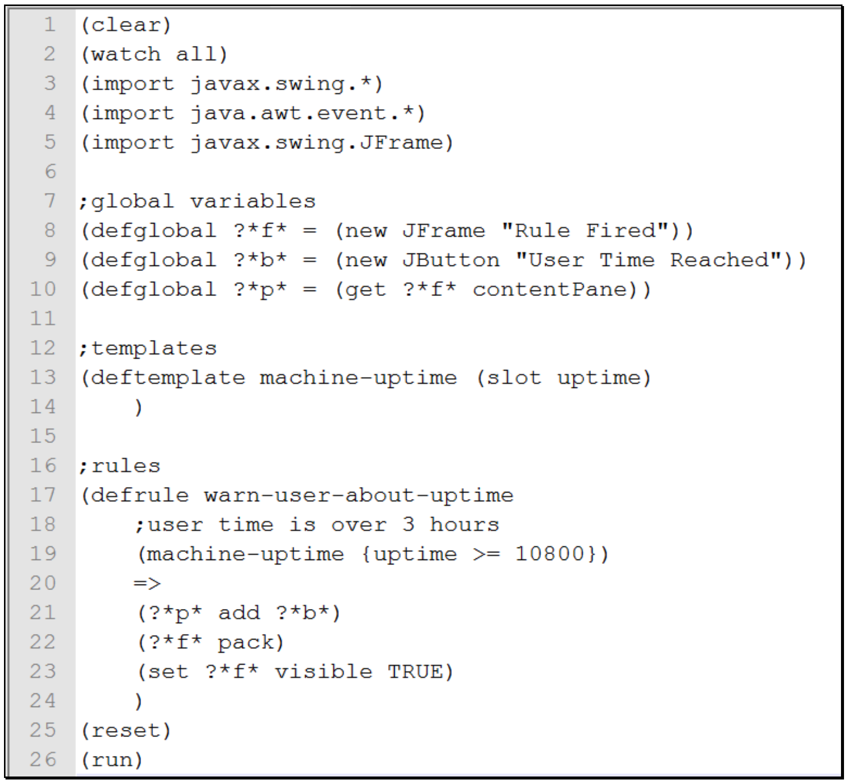
Figure 61: Conversion and Translation Performed by JADE (adapted from [43]**).**

Any agent that is registered to the agent platform can search for recipients in the AMS and sends applicable messages/requests using the *Agent.send()* method. Any agent willing to receive a message calls the *Agent.receive*() and *block()* methods in its cyclic behaviour. The *block()*

method tells the agent behaviour to wait for incoming messages and to process them when they arrive. The process of sending and receiving messages is scheduled as independent agent activities through the ticker behaviour and cyclic behaviour. The system agents share messages based on specific topics thereby allowing only agents interested in that topic to process the message. This review shows a fully functional distributed system with full system agents' negotiation support.

5.4. Rule Engine Test

This section reviews the rule engine that is embedded in the system agents. The rule engine offers the reasoning capabilities about the device state to the agents and performs basic management functions. The rules engine is supposed to check the data storage for the facts about the applications running, data consumption and system uptime and then when facts are true, the rule engine has to execute the rules. Figure 62 shows the system uptime rule and Figure 46 shows how to retrieve facts from the database. The rule engine retrieves the system uptime information from the *NodeInfo* table and compares it to the *machine-uptime* rule, and then when the user has reached three hours of usage a JFrame GUI is popped to the user monitor.



```
(clear)
(watch all)
(import javax.swing.*)
(import java.awt.event.*)
(import javax.swing.JFrame)

;global variables
(defglobal ?*f* = (new JFrame "Rule Fired"))
(defglobal ?*b* = (new JButton "User Time Reached"))
(defglobal ?*p* = (get ?*f* contentPane))

;templates
(deftemplate machine-uptime (slot uptime)
    )

;rules
(defrule warn-user-about-uptime
    ;user time is over 3 hours
    (machine-uptime {uptime >= 10800})
    =>
    (?*p* add ?*b*)
    (?*f* pack)
    (set ?*f* visible TRUE)
    )
(reset)
(run)
```

Figure 62: System Uptime Rule Test.

Figure 63 shows a GUI that displays to a user monitor when they reach 3 hours of system usage, the system uses Hyperic-Sigar library to get this information.

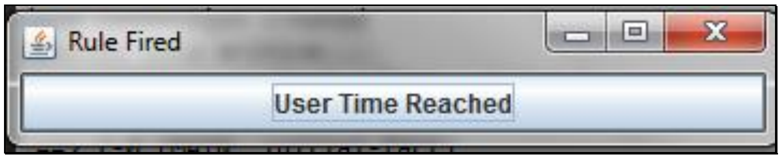


Figure 63: GUI Showing When System-Uptime Rule Becomes True.

When the rule fires an administrator gets their report notifying them about *machine-uptime* rule, Figure 64.

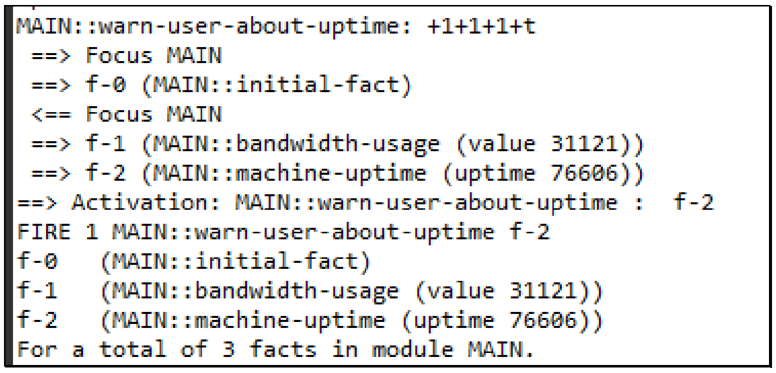
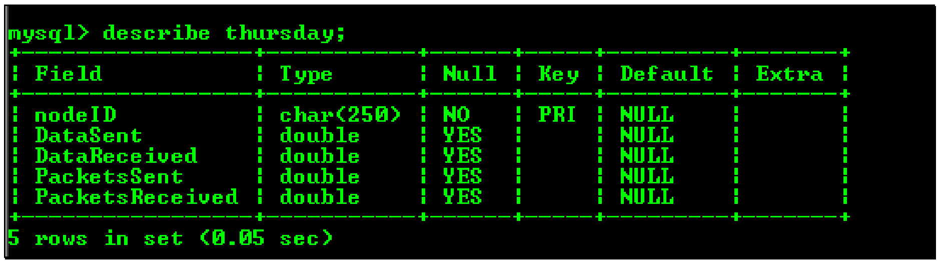

```
MAIN::warn-user-about-uptime: +1+1+1+t
 ==> Focus MAIN
 ==> f-0 (MAIN::initial-fact)
 <== Focus MAIN
 ==> f-1 (MAIN::bandwidth-usage (value 31121))
 ==> f-2 (MAIN::machine-uptime (uptime 76606))
==> Activation: MAIN::warn-user-about-uptime :  f-2
FIRE 1 MAIN::warn-user-about-uptime f-2
f-0   (MAIN::initial-fact)
f-1   (MAIN::bandwidth-usage (value 31121))
f-2   (MAIN::machine-uptime (uptime 76606))
For a total of 3 facts in module MAIN.
```

Figure 64: Terminal Report for an Administrator.

This shows that our rule engine is able to reason about the managed device state and executes the given rules when thresholds are reached.

5.5. Data Storage and Analysis

The *MasterAgent* consists of nine tables and the *NodeAgent* is made up of two tables only. These tables store network statistics and device information. The *NodeAgent* sends the network statistics and the device information that it resides on the *MasterAgent* for archiving and analysis. The *NodeInfo* table (Figure 31) stores the device information, the *Netstats* table stores network statistics information, *Monday, Tuesday, Wednesday, Thursday, Friday, Saturday* and *Sunday* tables stores the archived data consumption over a week. Figure 37 shows the information of interest acquired from a device that is: sent/received bytes in MB, sent/received packets and sent/received packet errors. The weekdays tables are described as in Figure 65, the table archives network data consumption for a day and is viewed in the *MasterAgent* by the GUI shown at Figure 57 and Figure 58.

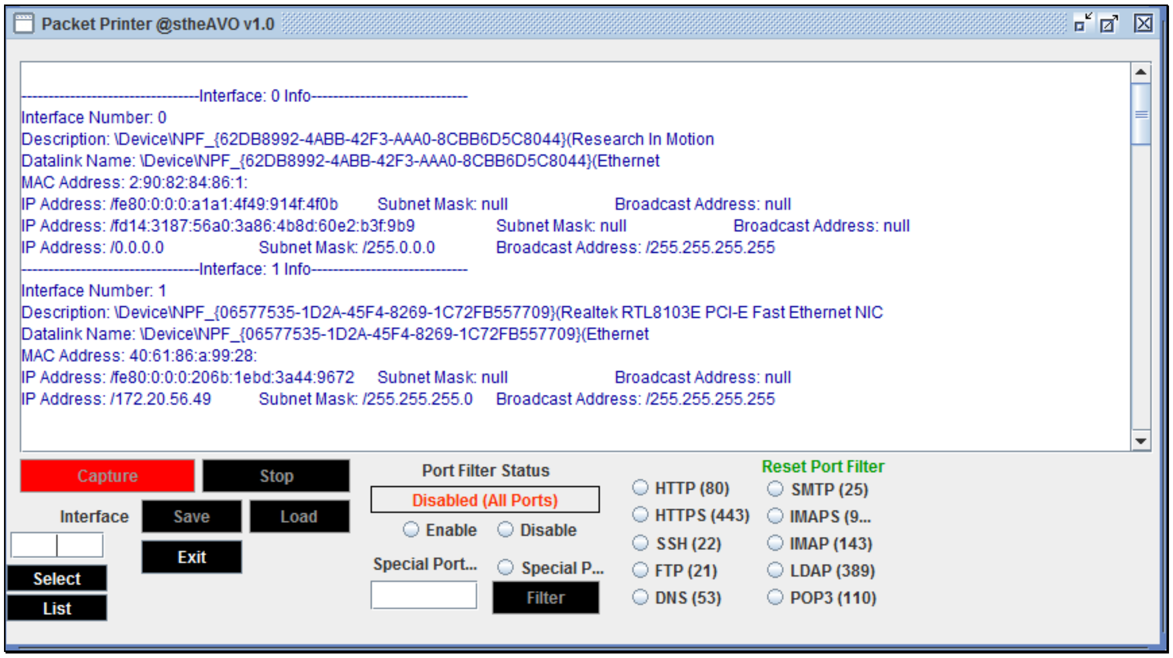


```
mysql> describe thursday;
+-----------------+-----------+------+-----+---------+-------+
| Field           | Type      | Null | Key | Default | Extra |
+-----------------+-----------+------+-----+---------+-------+
| nodeID          | char(250) | NO   | PRI | NULL    |       |
| DataSent        | double    | YES  |     | NULL    |       |
| DataReceived    | double    | YES  |     | NULL    |       |
| PacketsSent     | double    | YES  |     | NULL    |       |
| PacketsReceived | double    | YES  |     | NULL    |       |
+-----------------+-----------+------+-----+---------+-------+
5 rows in set (0.05 sec)
```

Figure 65: Fields of Week Days Table.

The *NodeAgents* logs in to the *MasterAgent* database using the IP address of the device where the *MasterAgent* resides, database name *DeviceInfo* and the database password. The *NodeAgents* on the network update the *MasterAgent* database with the device user and network information periodically. The week days tables are updated after each day, the user information table is updated when there is a new user on a network device, the hardware info is static so there is no need to update it every time and the network information is also updated with the new user on the device because the IP addresses change as most networks use Dynamic Host Configuration Protocol (DHCP) address leasing. The *NodeAgent* uses the ticker behaviour to facilitate the periodic database updates.

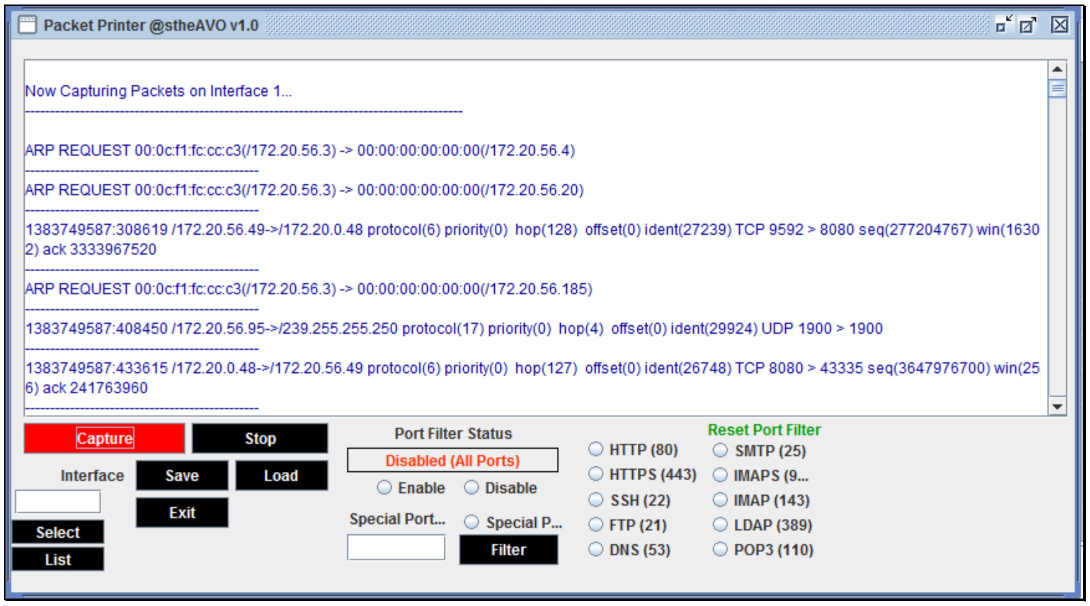
An administrator uses the user interface on the *MasterAgent* to view the network analysis. These interfaces provide a detailed device, network and user information for an administrator and they are started and reside on the device that has an agent platform. An administrator also has a GUI that shows network packets in real-time, the packet user interface lists the available network interfaces and an administrator chooses from the list which interface they want to sniff and they start the capturing process. The network packet shows the device communications information because every packet has source/destination MAC and IP addresses, IP Protocol and packet length. At first the packet capture GUI, Figure 35, shows disabled buttons except for *List* and *Exit*, because at first a user can only list network interface before starting the capture process or exit the whole idea. Figure 66 shows the list of network interface available on the device after a user has clicked on *List* button.



```
--------------------------------Interface: 0 Info----------------------------
Interface Number: 0
Description: \Device\NPF_{62DB8992-4ABB-42F3-AAA0-8CBB6D5C8044}(Research In Motion
Datalink Name: \Device\NPF_{62DB8992-4ABB-42F3-AAA0-8CBB6D5C8044}(Ethernet
MAC Address: 2:90:82:84:86:1:
IP Address: /fe80:0:0:0:a1a1:4f49:914f:4f0b       Subnet Mask: null               Broadcast Address: null
IP Address: /fd14:3187:56a0:3a86:4b8d:60e2:b3f:9b9            Subnet Mask: null               Broadcast Address: null
IP Address: /0.0.0.0               Subnet Mask: /255.0.0.0         Broadcast Address: /255.255.255.255
--------------------------------Interface: 1 Info----------------------------
Interface Number: 1
Description: \Device\NPF_{06577535-1D2A-45F4-8269-1C72FB557709}(Realtek RTL8103E PCI-E Fast Ethernet NIC
Datalink Name: \Device\NPF_{06577535-1D2A-45F4-8269-1C72FB557709}(Ethernet
MAC Address: 40:61:86:a:99:28:
IP Address: /fe80:0:0:0:206b:1ebd:3a44:9672    Subnet Mask: null               Broadcast Address: null
IP Address: /172.20.56.49          Subnet Mask: /255.255.255.0    Broadcast Address: /255.255.255.255
```

Figure 66: Packet Capture GUI Listing Network Interfaces.

After listing device network interfaces, the *Select* button becomes enabled and the cursor is focused on *Interface* text box for a user to type in the interface number inside the text box. When a user types the interface number and clicks the *Select* button the *Capture*, *Save*, *Stop* and *Load* buttons become enabled. When a user decides to capture packet on *Interface 1* which is an Ethernet interface, the GUI appends the network packets on GUI as in Figure 67.

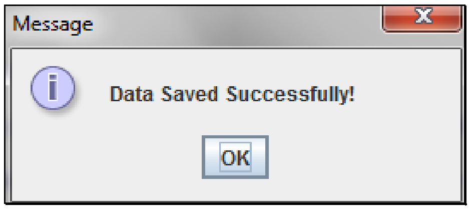


```
Now Capturing Packets on Interface 1...
--------------------------------------------------------------------------------

ARP REQUEST 00:0c:f1:fc:cc:c3(/172.20.56.3) -> 00:00:00:00:00:00(/172.20.56.4)
-------------------------------------------
ARP REQUEST 00:0c:f1:fc:cc:c3(/172.20.56.3) -> 00:00:00:00:00:00(/172.20.56.20)
-------------------------------------------
1383749587:308619 /172.20.56.49->/172.20.0.48 protocol(6) priority(0)  hop(128)  offset(0) ident(27239) TCP 9592 > 8080 seq(277204767) win(1630
2) ack 3333967520
-------------------------------------------
ARP REQUEST 00:0c:f1:fc:cc:c3(/172.20.56.3) -> 00:00:00:00:00:00(/172.20.56.185)
-------------------------------------------
1383749587:408450 /172.20.56.95->/239.255.255.250 protocol(17) priority(0)  hop(4)  offset(0) ident(29924) UDP 1900 > 1900
-------------------------------------------
1383749587:433615 /172.20.0.48->/172.20.56.49 protocol(6) priority(0)  hop(127)  offset(0) ident(26748) TCP 8080 > 43335 seq(3647976700) win(25
6) ack 241763960
-------------------------------------------
```

Port Filter Status: Disabled (All Ports)

Figure 67: Packet Capturing Process on Selected Interface.

When a user decides to save or load the packets to or from a file for analysis, they click on the *Save* and *Load* buttons and are presented with the following message when saving to file is successful, Figure 68.



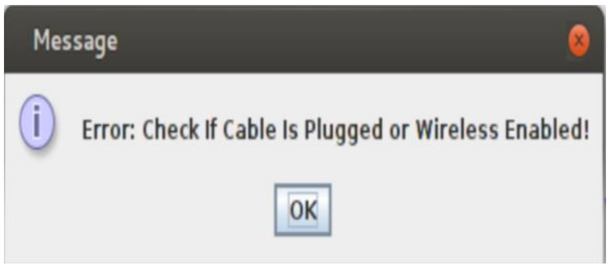
Data Saved Successfully!

Figure 68: Success on Saving Packets.

All this information is collected so the system agents and the administrator can analyze the network statistics and user operations on the network.

5.6. Network and Internet Connection Test

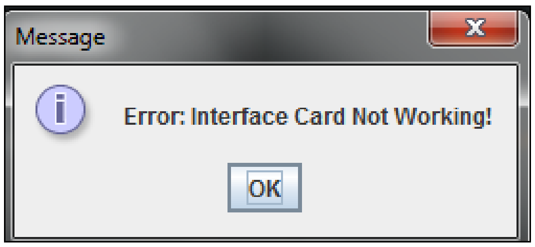
This section reviews how the *checkInternet()* and *checkInterface()* methods deal with the NIC and network and Internet failures. The test plan for the *checkInterface()* method is to observe how the *NodeAgent* reacts when the network cable is not connected or when the network adaptors are disabled or when the NIC is totally not working. When the network cable is unplugged or when the adaptors are disconnected, the following message is displayed on the user screen by the *NodeAgent* behaviour, Figure 69.


Message

Error: Check If Cable Is Plugged or Wireless Enabled!

Figure 69: Test Error for Network Cable or Wireless Adaptor.

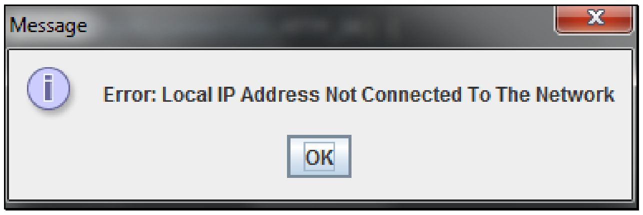
When the *NodeAgent* detects that the available NIC is not working, it displays the following message, Figure 70.


Message

Error: Interface Card Not Working!

Figure 70: Test for NIC.

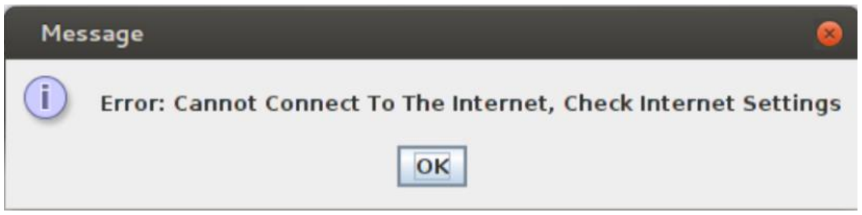
The *NodeAgent* tests the network connection by retrieving the IP address of the device and tests if it is not reachable by using the Java *isReachable()* method, Figure 42. When the agent cannot reach the local IP address, it displays a message informing the user that they are no longer connected to network and this could only arise if the network server is down, Figure 71.



Error: Local IP Address Not Connected To The Network

Figure 71: Testing Local IP Address Connectivity to the Network.

The user cannot have Internet access when the network device IP address is not connected to the network and/or when the server is temporally down. When the user is connected to the network, the *NodeAgent* sends a request to an external known site to check if the whole network has Internet access. When the connection refuses, the *NodeAgent* generates a message about the failure, Figure 72.



Error: Cannot Connect To The Internet, Check Internet Settings

Figure 72: Internet Connection Test.

These messages help the user to know where the problem is before they could call for assistance from the administrator. The *NodeAgent* performs this troubleshooting to identify what kind of an error is there, thereby eliminating the hustle from the users and ensuring high QoS by finding the failure before the users experience the problem.

5.7. Discussion

This chapter reviewed an end-to-end functionality of the system based on the detailed functional specifications. The purpose for this was to show how the system functions and to show that every module of the system, when integrated, functions correctly as whole. The system consists of the *NodeAgent* and *MasterAgent* agents, the *NodeAgent* is made of five modules which are:

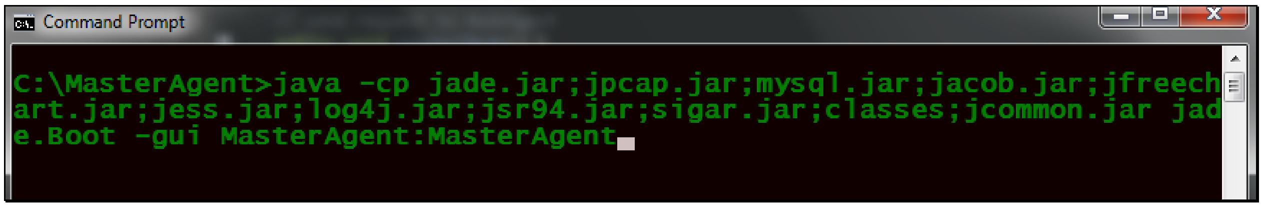
- System and User information collector;
- Inference Engine;
- Network statistics collector;
- Network and Internet Connection; and
- Ontology module.

NodeAgent is also characterized by a number of behaviours that automate the modules and communication computations, while the *MasterAgent* consists of five modules that are:

- Bandwidth meter;
- Packet capture;
- Consumed Data comparison;
- Web-interface; and
- Inference Engine.

It also has agent behaviours that facilitate the agent communication and module computations. Further, the *MasterAgent* has a database that stores the entire network and managed device information while the *NodeAgent* has a data storage that stores the information of the device it resides on. The *NodeAgent* resides on the managed devices and the MasterAgent resides on the agent platform, therefore making a distributed system. The number of *NodeAgents* that can be added to the agent platform is limitless thereby making the system to be flexible and cope with the scalability as the number of managed devices is increased.

The system has only one running instance of the main-container in an agent platform that is started by running the *MasterAgent* class and a limitless number of containers (that run *NodeAgents*) can be added to the main-container. Figure 47 shows the initialization of the main-container that consists of the *MasterAgent*, DF, AMS, RMA and tools to manipulate agents in the agent platform for example the *SnifferAgent* and *DummyAgent* with options to start new agents, kill running agents, clone/migrate agents, pause/resume agents. Figure 54 shows how the *NodeAgent* is started in a network device and added to the running agent platform. The *NodeAgent* is started without the *–gui* option on the terminal so that it would not start the RMA GUI or any other GUI (Figure 54), as it would inconvenience the device user and have a negative impact on the system performance [43]. The behaviours defined inside the agent start to perform their duties as soon as the *NodeAgent* and *MasterAgent* are initiated. For the system agents to operate there should be at least one main-container instance running in the agent platform at all times. Figure 73 shows how the main-container of the agent platform is started on the command line through the *MasterAgent* class.



```
C:\MasterAgent>java -cp jade.jar;jpcap.jar;mysql.jar;jacob.jar;jfreech
art.jar;jess.jar;log4j.jar;jsr94.jar;sigar.jar;classes;jcommon.jar jad
e.Boot -gui MasterAgent:MasterAgent
```

Figure 73: Starting the Main-Container.

The command includes the *-gui* option to start the RMA GUI with all the libraries and classes required by the *MasterAgent*.

This system provides lightweight footprint agents that can be installed in any platform regardless of the machine OS that supports JVM, storage capacity (as the agents are very small in size), power capacity and processing capability as the agents do not use much of the CPU. The system provides ad hoc alerting service to the administrator and system users and a web interface that provides network and node statistics. This improves the quality of service and helps the network users with specific knowledge about the network and node failures should they arise. The system also provides system agents management GUI, this allows network administrator to start new or clone and kill agents, view system agents communications and requests, and view system agents operations and function on the network.

5.8. Conclusion

This chapter served to test the system components functionality and showed that when integrated, they work as one. The main aim of this chapter was to show that the system agents can perform as described and can operate in any operating system. The tests that were done include performance management, agent communication, inference engine and data storage and analysis tests. The next chapter provides the study conclusion and discusses the future work.

6. Conclusion and Future Work

This is the last chapter of this dissertation; it provides an overall discussion on research findings and discusses how the research objectives were addressed.

6.1. Introduction

Chapter 5 provided the different system functionality tests to show how the system works and to address the requirements specified in Chapter 3. This dissertation has discussed the design and implementation details of the system and this chapter gives an overall summary of the dissertation. This chapter highlights how the research objectives were met and discusses the proposed future extensions of the system. The last section of this chapter gives an overall conclusion to the dissertation.

6.2. Dissertation Discussion

This dissertation presented an experimental study of a network management system using intelligent agents that can be used in small to medium sized networks. The use of intelligent agents in network management has been proven to be better than the traditional static management methods. This research discussed the main advantages and benefits of intelligent agents for network management. The basic motivation of this research was to develop a decentralized and flexible management system for SLL network thereby improving the QoS required by the users and ICTs deployed in the community. Developing this knowledge-based system required a lot of technologies to be integrated together and this was achieved through a thorough literature survey on network management systems based on intelligent agents. The research methodology followed the process of KE methodology to acquire knowledge about the implementation of the system and used the iterative development model as a life cycle for the system implementation. The iterative development model ensured that the system requirements were conformed through unit testing of the system prototypes.

The system was implemented using Java, JESS and MySQL languages on Windows and Ubuntu platforms; and it was tested on Windows and on UNIX systems. This was to ensure that it is platform independent as SLL mostly uses EduBuntu/Ubuntu and few devices run on Windows OS. The system package uses a number of libraries to make its operations successful for example JADE framework is used to implement the agent platform that complies with FIPA specifications; Jpcap is used to capture network data and packets; Hyperic-Sigar is used to collect the system and user information, JFreeChart is used to draw a dynamic chart for the presentation of network data; MySQL is used to facilitate the storage of network, system and user information in database and JESS is used to implement the inference engine. The system consists of two types of agents: the *MasterAgent* that resides on the agent platform and the *NodeAgent* that is designed to reside on the network devices. The system is designed to have as many instances of the *NodeAgent* as the network devices and only one or two instances of the *MasterAgent*. The second *MasterAgent* would be installed so the system can have a redundant agent platform in case the main agent platform dies unexpectedly. In this case, for the second agent platform to know about the system agents operations it would require to be linked to the DF of the main agent platform. This will ensure that it knows the agents operating on the network and the services they offer on the network management goal.

6.3. Discussion on Research Objectives

To develop this system a detailed literature survey was conducted so the system requirements can be specified and identify the technologies that will be required to achieve the main goals of this study. The basic aim of this study was to develop an intelligent and flexible network management system. The research objectives were answered as follows:

Objective 1: a thorough survey was performed on traditional network management systems and on agent-based management systems with their advantages and disadvantages.

Objective 2: the JADE platform integrated with JESS rule engine were chosen to develop decentralized system agents.

Objective 3: a decentralized network management system with the following features was developed:

- Network monitoring user interfaces that provide network statistics; these interfaces show the data consumed by the users on the network;
- Data storage for the inventory of network devices, users on the network and network statistics;
- Web interface that shows network users and device hardware information;
- Agent platform that creates agent execution environment and also facilitates the decentralization of agents in the network;
- Asynchronous system agent's communication and agents that reside on network devices. This minimizes the bandwidth utilization when there are management functions to be carried on the managed device, now these operations can happen without the concern of the central server/controller; and
- Knowledge-based system that uses the information in hand (facts) and the goals to take decisions about user operations on the network.
- **Objective 4**: experimental lab tests were done on the system to test if the system complies with the functional requirements described in Section 3.4.1.

This shows the success of developing a decentralized network management based on intelligent agents because the system conforms to the system requirements specified and to the objectives of this research.

6.4. Limitations and Future Work

Even though the system functions positively (based on the functional tests), to install the system agents there is a need of an above average computer literacy. This can be improved by creating an executable file of system agents. This will also allow the agents to be started easily when the network nodes starts. All the tests performed on the system implemented we done in the lab environment, a need to deploy this autonomous system in a computer network system is necessary. A possible extension to the system will be to develop a mobile application for a network manager to view network statistics and be able to interact with the agents. The current system is a stand-alone application. In future, the system can be integrated with other intelligent systems hence there will be a need to find a mechanism for that integration.

6.5. Overall Conclusion

This chapter described the overall discussion on the dissertation and discussed the functionality offered by the system with the suggested extensions to the research. This dissertation presented the development of a distributed network management using software agents. Most importantly, it highlighted the advantages and the benefits of using software agents in management systems with their drawbacks as compared to other competing technologies. Various mechanisms for network management were discussed in this study with their advantages and limitations. The study adopted used several tests for the technical system operation and showed how the agent platform and the system agents are started on the network devices. The tests showed that the system is capable of monitoring and performing basic management on network devices. Further, it proved that the system is platform independent and meets the system requirements.

7. References

[1] E. B. Parker, “Closing the digital divide in Rural America,” *Telecommunications Policy*, vol. 24, no. 4, pp. 281–290, 2000.

[2] A. Goldstein and D. O. Connor, “e-Commerce for Development: Procpects and Policy Issues,” *Paris: OECD Development Centre*, vol. Vol. 2001, 2000.

[3] A. Dhananjay, M. Tierney, J. Li, and L. Subramanian, “WiRE : A New Rural Connectivity Paradigm,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 462–463, 2011.

[4] J. Ahn, “Fault-tolerant Mobile Agent-based Monitoring Mechanism for Highly Dynamic Distributed Networks,” *IJCSI International Journal of Computer Science*, vol. 7, no. 3, pp. 1–7, 2010.

[5] M. L. Griss, “Software Agents as Next Generation Software Components,” *Components-Based Software Engineering: Putting the Pieces Together.*, pp. 1–11, 2001.

[6] H. Kim and N. Feamster, “Improving Network Management with Software Defined Networking,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[7] I. Cisco Systems, *Network Management Basics, Fourth Edition*. Cisco press, 2004, pp. 87–89.

[8] B. Aboba, J. Arkho, and D. Harrington, “RFC 2975: Introduction to Accounting Management,” *The Internet Society*, no. October, 2000.

[9] L. E. Miller, J. J. Kelleher, and L. Wong, “Evaluation Of Network Reliability,” *White Paper*, no. Report, December 2004, 2004.

[10] M. Kundan, *OSS for Telecom Networks: An Introduction to Networks Management*. Springer, 2004, pp. 133–135.

[11] A. Goldstuck, “Internet Matters : The Quiet Engine of the South African Economy,” *World Wide Worx*, 2012.

[12] I. Siebörger and A. Terzoli, “Network Provision for Rural Development: The Case of the Siyakhula Living Lab,” *Unpublished Manuscript*, pp. 1–6, 2012.

[13] A. M. Elmahalawy, “Intelligent Agents and Multi-Agent Systems,” *Journal of*

Engineering and Technology, vol. 2, no. 1, 2009.

[14] J. M. Bradshaw, “An Introduction to Software Agents,” *MIT Press*, 1997.

[15] R. Sugar and S. Imre, “Dynamic Agent Domains in Mobile Agent Based Network Management,” *Networking—ICN 2001. Springer Berlin Heidelberg*, pp. 468–477, 2001.

[16] Y. Kim and S. Hariri, “ExNet : An Intelligent Network Management System,” *WebNet*, pp. 0–5, 1998.

[17] OpenNMS, “Online: http://www.opennms.org/,” 2013.

[18] D. Hustace, “End To End Monitoring: Hyperic HQ Intergration with OpenNMs,” *White Paper, The OpenNMS Group Inc.*, 2013.

[19] H. Wang and Y. Chen., “Network topology description and visualization,” *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference*, vol. 6, IEEE 20, no. February, 2010.

[20] OpManager, “Online: http://www.manageengine.com/network-performance-management.html/,” 2013.

[21] S. M. Magda, A. B. Rus, and V. Dobrota, “Nagios-based network management for Android, Windows and Fedora Core terminals using Net-SNMP agents,” in *Roedunet International Conference (RoEduNet), 2013 11th*, 2013, pp. 1–6.

[22] Nagios, “Online: http://www. nagios.org,” 2013.

[23] Hyperic HQ, “Hyperic HQ: Online http://www.hyperic.com/,” vol. Accessed J, 2013.

[24] C. Thomas, “GroundWork Monitor Architecture Overview,” *GroundWork Open Source, Online: http://www.groundworkopensource.com/*, 2013.

[25] Argus, “Online: http://www.argussoftware.com/en/,” 2013.

[26] Cacti, “Online: http://www.cacti.net/,” 2013.

[27] SNMPc, “Online: http://www.snmpc.ca/,” 2013.

[28] NetXMS, “Online: http://www.netxms.org,” 2013.

[29] G. Goldszmidt, Y. Yemini, and S. Yemini, “Network Management by Delegation: The MAD Approach,” *Communications Magazine, IEEE*, vol. 38, no. 3, pp. 66–70, 1998.

[30] V. Ioannis, B. Nick, S. Ilias, M. Martin, O. Sascha, F. Ivan, P. Zoltán, S. Janos, V. Igor, Y. Sergey, and N. Igor, “ExperNet : An Intelligent Multi-Agent System for WAN Management ∗,” *Intelligent Systems, IEEE.*, vol. 17, no. 1, pp. 62–72, 2002.

[31] M. Naylor, “The Use of Mobile Agents in Network Management Applications,” *Submitted in partial fulfilment of the requirements of Napier University, for the degree of MSc Information Technology (Systems Integration). School of Computing (2000).*, no. January, 2000.

[32] A. Maj, J. Jurowicz, J. Koźlak, and K. Cetnarowicz, “A Multi-Agent System for Dynamic Network Reconfiguration,” *Multi-Agent Systems and Applications III. Springer Berlin Heidelberg*, pp. 511–521, 2003.

[33] Netview, “Online: http://www.netview.com/,” 2013.

[34] C.-M. Chen and C.-P. Wei, “Efficient Network Monitoring for Large Networks,” *Journal of Computers, Proceeding of 2007*, vol. 18, no. 4, 2008.

[35] J. Lefebvre, S. Chamberland, and P. Samuel, “A Network Management Framework Using Mobile Agents,” *Electrical and Computer Engineering, IEEE CCECE 2003.*, vol. 2, pp. 737–740, 2003.

[36] D. Mitrovic, Z. Budimac, M. Ivanovic, and M. Vidakovic, “Improving Fault-Tolerance of Distributed Multi-Agent Systems with Mobile Network-Management Agents.,” *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, vol. 5, pp. 217–222, 2010.

[37] A. Tripathi, M. Koka, S. Karanth, I. Osipkov, H. Talkad, T. Ahmed, D. Johnson, and S. Dier, “Robustness and security in a mobile-agent based network monitoring system,” *International Conference on Autonomic Computing, 2004. Proceedings.*, pp. 320–321, 2004.

[38] V. Pere, M. Jose L., A. Bueno, E. Calle, and F. Lluis, “Distributed Network Resource Management using a Multi-Agent System: Scalability Evaluation.,” *International Symposium on Performance Evaluation of Computer and Telecommunication Systems.*, vol. SPECTS'04, pp. 355–362, 2004.

[39] S. Papavassiliou, A. Puliafito, T. Orazio, and J. Ye, “Mobile Agent Based Approach for Efficient Network Management and Resource Allocation : Framework and Applications.,” *Selected Areas in Communications, IEEE JOurnal.*, vol. 20, no. 4, 2002.

[40] R. Schoonderwoerd, O. Holland, and J. Bruten, “Ant-like agents fschor load balancing in telecommunications networks,” *Proceedings of the first international conference on*

Autonomous agents - AGENTS ’97, pp. 209–216, 1997.

[41] I. Satoh, “Building Reusable Mobile Agents for Network Management,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions.*, vol. 33, no. 3, pp. 350–357, 2003.

[42] N. R. Jennings and M. J. Wooldridge, *Agent Technology: Foundations, Applications and Markets*. London, UK: Springer Berlin / Heidelberg, 1998, pp. 3–49.

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

[44] S. Kraus, “Negotiation and Cooperation in Multi-Agent Environments *,” *Aritificial Intelligence*, vol. 94, no. 1, pp. 79–97, 1997.

[45] B. Hu, J. Hidders, and P. Cimiano, “A rule engine for relevance assessment in a contextualized information delivery system,” *Proceedings of the 15th international conference on Intelligent user interfaces - IUI ’11*, pp. 343–346, 2011.

[46] OpenRules, “Online: http://www.open.com/,” 2013.

[47] E. J. Friedman-Hill, “Jess , The Expert System Shell for the Java Platform,” *Sandia National Laboratories*, no. November 2008, 2008.

[48] Prolog, “Online: http://http://www.amzi.com/ExpertSystemsInProlog/02usingprolog.htm/,” 2013.

[49] D. Merritt, *Building Expert Systems in Prolog*. New York, NY: Springer New York, 2001, pp. 9–11.

[50] D. Gavalas, G. E. Tsekouras, and C. Anagnostopoulos, “A mobile agent platform for distributed network and systems management,” *Journal of Systems and Software*, vol. 82, no. 2, pp. 355–371, Feb. 2009.

[51] G. Cabri, L. Leonardi, and F. Zambonelli, “Weak and Strong Mobility in Mobile Agent Applications,” *Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java, Manchester, UK*, 2001.

[52] C. Bădică, Z. Budimac, H.-D. Burkhard, and M. Ivanovic, “Software agents: Languages, tools, platforms,” *Computer Science and Information Systems/ComSIS*, vol. 8, no. 2, pp. 255–298, 2011.

[53] M. T. Kone, A. Shimazu, and T. Nakajima, “The State of the Art in Agent

Communication," *Knowledge and Information Systems*, vol. 2, pp. 259–284, 2000.

[54] M. Barbuceanu and M. S. Fox, "COOL : A Language for Describing Coordination in Multi Agent Systems," *ICMAS*, pp. 17–24, 1995.

[55] M. Dastani, A. El Fallah-seghrouchni, A. Ricci, and M. Winikoff, *Programming Multi-Agent Systems*. Springer, 2008, 2007.

[56] R. H. Bordini, L. Braubach, J. J. Gomez-sanz, G. O. Hare, A. Pokahr, and A. Ricci, "A Survey of Programming Languages and Platforms for Multi-Agent Systems," *Informatica (Slovenia)*, vol. 30, no. 1, pp. 33–44, 2006.

[57] M. Dastani, M. B. Van Riemsdijk, and J. Meyer, "Programming Multi-Agent Systems in 3APl," *Multi-agent programming. Springer US*, pp. 39–67, 2005.

[58] M. Thielscher, "FLUX : A Logic Programming Method for Reasoning Agents," *Theory and Practice of Logic Programming*, vol. 5., no. 4–5, pp. 533–565, 2005.

[59] C. A. Iglesias, M. Garijo, and J. C. Gonzalez, "A Survey of Agent-Oriented Methodologies," *Framework 2*, vol. 34, 1999.

[60] O. Arazy and C. C. Woo, "Analysis and Design of Agent-Oriented Information Systems," *The knowledge engineering review*, vol. 17, no. 3, pp. 215–260, 2002.

[61] F. Bergenti and P. Turci, *Agent-Oriented Software Engineering*, Vol. 11. Springer1, 2004, pp. 65–147.

[62] L. Cernuzzi, T. Juan, L. Sterling, and F. Zambonelli, "The Gaia Methodology- Basic Concepts and Extensions," *Methodologies and Software Engineering for Agent Systems.*, pp. 69–87, 2004.

[63] P. Moraïtis, E. Petraki, and N. I. Spanoudakis, "Engineering JADE Agents with the Gaia Methodology," *Agent Technologies, Infrastructures, Tools, and Applications for e-Services.*, pp. 77–91, 2003.

[64] M. Wooldridge, N. R. Jennings, and D. Kinny, "A Methodology for Agent-Oriented Analysis and Design," *Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 3, pp. 85–312, 2000.

[65] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236., 2004.

[66] G. Picard and M. Gleizes, “The Adelfe Methodology: Designing Adaptive Cooperative Multi-Agent Systems,” *Methodologies and Software Engineering for Agent Systems.*, pp. 157–176, 2004.

[67] C. Bernon, M. Gleizes, S. Peyruqueou, and G. Picard, “ADELFE , a Methodology for Adaptive Multi-Agent Systems Engineering,” *In Engineering Societies in the Agents World III*, pp. 156–169, 2003.

[68] L. Padgham and M. Winikoff, “Prometheus : A Methodology for Developing Intelligent Agents,” *Agent-oriented software engineering III*, pp. 174–185, 2003.

[69] T. S. Vaquero, J. R. Silva, and J. C. Beck, “A brief review of tools and methods for knowledge engineering for planning & scheduling.,” *Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling, Germany*, pp. 7–14, 2011.

[70] N. Perry and S. Ammar-Khodja, “A Knowledge Engineering Method for New Product Development,” *Journal of Decision Systems*, vol. 19, no. 1, pp. 117–133, Jan. 2010.

[71] M. Garijo, A. Cancer, and J. J. Sánchez, “A Multiagent System for Cooperative Network-Fault Management,” *Proceedings of the First International Conference on the Practical Applications of Intelligent Agents and Multi-agent Technology, PAAM96.*, no. 1, pp. 279–294, 1996.

[72] W. Lau, G. F. Rosenbaum, and S. Jha, “Comments on ‘Dynamic Routing of Restorable Bandwidth-Guaranteed Tunnels Using Aggregated Network Resource Usage Information’,” *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 244–245, 2008.