INTELLIGENT MOBILE AGENT APPLICATION FOR SPECTRUM RESOURCE MANAGEMENT IN COGNITIVE RADIO NETWORKS

By

ERIC-NWONYE, NNENNA CHRISTINE
(STUDENT NUMBER: 23989696)

DISSERTATION SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE (MSc.) IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF MATHEMATICAL AND PHYSICAL SCIENCES
FACULTY OF AGRICULTURE, SCIENCE AND TECHNOLOGY
NORTH-WEST UNIVERSITY, MAFIKENG CAMPUS

SUPERVISOR: PROFESSOR O. O. EKABUA

NOVEMBER, 2012

DECLARATION

I declare that this Research Project on **Intelligent Mobile Agents Application for Spectrum Resource Management in Cognitive Radio Networks** is my work, and has never been presented for the award of any degree in any university. All the information used has been dully acknowledged both in text and in the references.

Signature _____ Date _____

Eric-Nwonye, Nnenna Christine

Approval

Signature _____ Date _____

Supervisor: **Prof O.O. Ekabua**
Department of Computer Science
Faculty of Agriculture, Science and Technology
North West University, Mafikeng Campus
South Africa

DEDICATION

This research project is dedicated to the three loves of my life:

Eric Maduabuchi Nwonye,

Nadal Chisom Nwonye

and

Eric Chibuike Nwonye.

ACKNOWLEDGEMENTS

I wish to first express my gratitude and praise to the Almighty God, for making everything possible. I thank Him for giving me life, knowledge, wisdom and seeing me through to the successful completion of this research project and my programme.

I am immensely grateful and thankful to Prof O.O. Ekabua, my supervisor, for his support, advice, discussions and useful criticism while carrying out this research. I am also thankful for his support and encouragement to me in pursuing this degree.

I am very thankful to the former Director of the School of Mathematics and Physical Sciences and now Acting Dean of the Faculty of Agriculture, Science and Technology, Prof Eno Ebenso for his support and encouragement.

I am also grateful to the lecturers and staff of the Department of Computer Science, North West University, Mafikeng Campus, for their help and support.

I also appreciate the help, support and encouragement of all my family and friends – Ferdinand & Irene Nwonye, Fanny Ekabua, Clara Achibiri, Ifeoma Ohaeri, Ijeoma Tasie Utah, Nosipho Ndladlu, Thuso Moemi, Michel Mbougni, Francis Lugayizi, Boyce Sigweni, Hope Tsholofelo Mogale, Ifeyinwa Anyikwa, and Olajide Johnson.

I am very grateful to my mother, Mrs Felicia Jonah, for her love, support and patience whilst undertaking this programme.

Finally, I wish to express my love and gratitude to my husband, Eric Maduabuchi Nwonye, for his love, support and encouragement. And to my boys, Nadal Chisom & Eric Chibuike, for inspiring me, even in the midst of their distractions. It would not have had any meaning without you boys. Thank you.

Abstract

With the increasing demand for wireless communication, efficient spectrum utilization has become a challenge. This is because wireless spectrum is a costly resource licensed by relevant agencies to operators for a specified purpose and for long periods of use, often running into decades. But because a large portion of these licensed spectrums are sporadically used, it results into underutilization of valuable frequency resource. Consequently, dynamic spectrum access technology is proposed to address this spectrum underutilization. This research shift focused on the development of cognitive radios to further improve spectrum efficiency. The fundamental objective of cognitive radio networks is for unlicensed users (also called cognitive or secondary users), to utilize an idle frequency band and vacate the band as soon as the licensed users are detected. This action imposes a great challenge to radios due to the high fluctuations experienced in the available spectrum, as well as diverse quality of service requirements. Consequently, to take this concept a step further, in this research paper, the use of intelligent mobile agents for spectrum resource management within the cognitive radio network was developed. This was achieved by injecting mobile agent codes into the network to detect, decide and distribute spectrum resource efficiently amongst the competing radios without interferences.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AMM	Agent Memory Module
CFP	Call For Proposal
CR	Cognitive Radio
CRN	Cognitive Radio Networks
DSA	Dynamic Spectrum Access
IMA	Intelligent Mobile Agent
IMACRN	Intelligent Mobile Agent for Cognitive Radio Networks
JVM	Java Virtual Machine
MA	Mobile Agent
MCD	Mobile Code Daemon
MCM	Mobile Code Manager
MF	Migration Facility
NC	Network Component
PU	Primary User
PUA	Primary User Agent
QoS	Quality of Service
SDA	Spectrum Decision Agent
SSA	Spectrum Sensing Agent
SU	Secondary User
SUA	Secondary User Agent
VMC	Virtual Managed Component
WSN	Wireless Sensor Network

Chapter 1

Introduction and Background

1.1 Overview

Cognitive radio technology is the key enabling technology for dynamic spectrum access (DSA), which makes it possible for unlicensed users to opportunistically share the frequency channel with the licensed user [1]. The idea of this technology was conceived due to the inefficient way the spectrum was being used, in addition to the increase in spectrum demand due to new applications. Wireless frequencies are assigned statically to specific users and for specified use, leading to simplicity and better service quality. The drawback of this static allocation is that it leads to very inefficient use of the spectrum because many allocated frequency bands are significantly underutilized. Cognitive radios can sense and react to their operating environment by dynamically adapting themselves for good application and network performance. These characteristics enable them to autonomously identify unused frequency bands and allow for usage of idle licensed frequency bands by unlicensed users. Cognitive radio networks are, therefore, composed of cognitive, spectrum-agile devices capable of changing their configurations on the fly based on the spectral environment [1,3,5].

However, as cognitive radio data and communication networks become more complex and distributed in nature, typical cognitive radio networks with control channels suffer from scalability and flexibility problems as it involves frequent signal and data transmission towards the control channel for processing. Secondly, there are multiple radios in a cognitive radio network, each vying to make use of the temporarily available spectrum hole. In this sense, a dynamic intelligent scalable resource distribution and management model based on intelligent mobile agents provides an attractive perspective to the lack of flexibility and scalability of such control channel systems.

Agents are programs that perform certain or specified tasks on behalf of the user [6]. They can be stationary (executes only on the system where it began execution) or mobile (can move to a remote system for execution. Mobile agents perform a user's task by migrating and executing on several hosts connected to the network. When an agent travels, it transports

its state and code with it. In this context, the agent state can be either its execution state, or the agent attribute values that help it determine what to do when it resumes execution at its destination [6,7].

Intelligent mobile agents are agents that have the ability to learn from their environment or from experience. They have the ability to operate without a user's involvement, ability to modify their behavior in response to changing environment and ability to make a decision. The main difference between an intelligent agent and traditional agent is that the former performs not only actions pre-specified by a user but also those necessitated by later changes in the environment [7].

A software agent is a self-contained computer program that is capable of controlling its own decision-making and acting based on its perception of its environment, in pursuit of one or more objectives. An agent possesses a number of attributes: acting on behalf of others, autonomy, ability to perceive reason and act in its environment, learn from experience and socially interact and communicate with other agents [17,36].

Mobile agents are a special category of software agents that can migrate through many nodes of a heterogeneous network of devices, under their own control, in order to perform tasks using resources of these nodes. A mobile agent travels from one node to another, performing tasks on behalf of its owner or originator. In the mobile agent paradigm, agents can move to the place where the data is stored, performing queries and filtering relevant information before sending this information to the client. In this context, it is cheaper to transport a small agent to the source of the data, than to bring the entire query result back to the node in order to be processed. Moreover, as the interaction between the agent and the resource (after moving) is performed in the same host, without the transmission of messages through the network, this paradigm is indicated for some kind of real-time applications.

Mobile agents introduce a new software and communication architecture, allowing a program to travel between machines for remote execution, even in heterogeneous cognitive radio networks. By transporting the agent code to the host machine in a distributed cognitive radio network, there is no need to bring intermediate signals and data across the network and

thus a significant amount of network bandwidth use and communication delay can be avoided [17,19].

Mobile agents allow the transformation of current networks into remotely programmable platforms. They can travel in the network, following their itinerary and carrying logic and data to perform a set of management tasks in order to meet their designed objectives. They are a powerful software interaction model that allows a program to be moved between hosts for remote execution and are the solution for managing distributed networks. The primary goal of using mobile agents in management of telecommunication networks is reducing network traffic through load balancing and building scalable and reliable distributed resource and network management systems [17,22]

Spectrum resource management and distribution among secondary users is an important issue in a cognitive radio network environment. Control channel-based network management frameworks suffer from problems such as interoperability, reliability, flexibility and scalability, as the cognitive radio networks become more geographically distributed [7]. In this research, an intelligent mobile agent-based framework for resource distribution and network management in a cognitive radio environment is developed. The intelligent mobile agent code is injected into the network to detect, decide and distribute spectrum resources efficiently amongst the radios in the network.

1.2 Background Study

Cognitive radio technology offers a new mechanism for flexible usage of the frequency spectrum. Even though the number of available frequency bands are decreasing, considerable portions of the frequency spectrum at a given time and place are not fully utilized. This spectrum under-utilization is caused by the fixed spectrum assignment policy used by the regulatory bodies in allocating spectrum to specific users and for specified use. Some of the advantages of fixed allocation of spectrum are [1,4,5]:

1. It is relatively easy to enforce policy once allocations have been made because there is no ambiguity about who can use the spectrum.
2. Giving exclusive licenses encourages investments in infrastructure, which ultimately benefits the society in the form of new services.
3. Dedicating frequency to specific uses simplifies equipment and deployment and often leads to better quality of service.

While fixed assignment of spectrum has many advantages, studies by the Federal Communication Commission indicate that many spectrum bands allocated through static assignment policies are used only in localized geographical areas or over limited period of time, and that the average utilization of such bands varies between 15% and 85% [1,4,5]. Although the intention of the regulatory bodies was to manage the spectrum, which is a valuable and finite resource (in a way it will benefit the society), such static allocation leads to inefficient use of the spectrum. One reason is that spectrum use is often localized, leaving many frequency bands unused in other areas. And being that most easily usable bands have been allocated, static allocation creates a spectrum shortage that hinders the growth of new wireless applications, thus necessitating the need to investigate more efficient spectrum allocation strategies. The limited available spectrum and the inefficiency in spectrum usage necessitate a new communication method to exploit the wireless frequency spectrum opportunistically [1,2,5].

Cognitive radios offer this dynamic use of the spectrum by allowing unlicensed users to operate in such under-utilized licensed frequency band in an intelligent way, without constraining the privileges of licensed or primary users. For example, the skipped channels in broadcast TV can be safely reused in nearby regions since TV only uses alternating channels in the spectrum, in any location. These unused TV channels, called white spaces, are an attractive target for dynamic spectrum access (DSA) because they operate at an easy to use frequency and their availability is fairly static and applies to a large area [1,4,5]. In may 2004, the FCC released a Notice of Proposed Rulemaking (NPRM 04 – 186) on "Unlicensed Operation in the TV Broadcast Bands", that proposes that certain unlicensed devices make use of TV white space [8,9].

A cognitive radio enabled secondary user is capable of periodically scanning and identifying available channels in the frequency spectrum. A channel is said to be available if a secondary user can transmit and receive messages on the channel for a reasonable amount of time, without interference to or from the primary users. Cognitive radios in a cognitive radio network therefore need to use the spectrum in such a manner as not to infringe on primary users and at the same time ensure maximum efficient management of the spectrum for each radio in the network [1,3].

Spectrum management in cognitive radio networks is a very important issue and entails four processes [3,4]:

i. **Spectrum Sensing or Detection:** in which cognitive radios monitor the channels and detect spectrum holes (determining the portions of the spectrum currently available).
ii. **Spectrum Decision:** selecting and allocating the best available channel based on spectrum availability and internal or external policies.
iii. **Spectrum Sharing:** coordinating access to this channel with other users, in order to prevent multiple users colliding in overlapping portions of the spectrum and for fair spectrum distribution.
iv. **Spectrum Mobility:** effectively vacating the channel when the licensed user is detected and continuing the communication in another vacant portion of the spectrum.

A challenge the cognitive radio networks face is that the performance of cognitive radio networks decreases sharply as the size of the network increases [2,3]. It is very difficult for radios in a network to easily maintain these management functionalities due to scalability and flexibility problems which networks suffer, thus necessitating the use of a dynamic and intelligent method of helping radios to manage spectrum resource [1,2,3]. An attractive method and solution is the use of intelligent mobile agents in a cognitive radio network.

Intelligent mobile agents (IMA) are autonomous programs situated within an environment (either a host or a network), which sense the environment and act upon it to achieve their goals. They complete their tasks by migrating from one device to another, sense the

environment and achieve their own local knowledge and experience. An Intelligent mobile agent has the following properties:

1. **Autonomy:** it can operate without the direct intervention of devices or humans.
2. **Decision Making:** it can make reactive and proactive decisions.
3. **Temporal Continuity:** its process runs continuously, either active in the foreground or passive in the background.
4. **Goal Oriented:** it is capable of handling a task to meet its desired goal.
5. **Mobility:** capable of roaming around in an electronic network.
6. **Communicative:** it can interact with other agents, devices and humans through some kind of agent communication language or a proxy.
7. **Collaborative:** it computes the desired tasks of the users or process by cooperating with other agents.
8. **Learning:** an agent can learn the environment and develop a certain degree of reasoning to take intelligent actions and decisions that improves the efficiency of the system.

The independence and mobility of intelligent mobile agents reduce bandwidth problems by moving a query from client to the server. It not only saves repetitive request or response handshake, it also addresses the problems created by intermittent and unreliable network connections since agents can easily work off-line and communicate their results when the application is back on-line [6,19,22,25].

By using intelligent mobile agents, the cognitive radio network can achieve a fair sharing of the spectrum and be able to recover quickly if the spectrum is reclaimed by the primary user. Intelligent mobile agent is an attractive solution to cognitive radio network resource management because their properties can help in achieving a balanced network operation and make the most out of the available radio resources in order to avoid interference.

1.3 Problem Statement

Radio frequency is a very important and finite resource. Its use in one area affects its availability in another area. The essence of cognitive radios is to efficiently utilize this resource by allowing unlicensed users to make use of an idle licensed spectrum to avoid spectrum wastage. There are multiple radios in a cognitive radio network, each vying to make use of the temporarily available spectrum hole which may result in collisions, starvation and bottleneck. Therefore, there is considerable interest in developing methods for efficient spectrum resource distribution and management among cognitive radios in a cognitive radio network. Furthermore, in a cognitive radio network, available spectrums may show different characteristics with the bandwidth, the primary user activity and acceptable interference limit, which affect both the sensing accuracy and efficiency. An efficient sensing technique is then essential in cognitive radio networks.

Intelligent mobile agents in cognitive radio networks can assist in resource management and distribution. The developed system, Intelligent Mobile Agent for Cognitive Radio Networks (IMACRN), is a mobile agent middleware that generates an intelligent mobile framework for deploying applications in cognitive radio networks. It is a cooperative multi-agent system, in which agents are deployed over primary and secondary user devices.

1.4 Research Questions

Against the background of the problem stated above, this research seeks to address the following research questions (RQs):

RQ1: How can the proper spectrum bands be selected in a cognitive radio network having multiple available spectrum bands with different channel characteristics?

RQ2: How can resource sharing be coordinated in a cognitive radio network to prevent multiple users colliding in overlapping portions of the spectrum?

RQ3: What method can be used for efficient spectrum mobility or spectrum reassignment?

1.5 Research Goal

The main goal of this research is to formulate a framework for efficient resource distribution and management in a cognitive radio network, using intelligent mobile agent codes injected in the network.

1.6 Research Objectives

In achieving the research goal, the following objectives shall be used:

i. Analyze current or existing literature on spectrum detection and distribution in cognitive radio networks.
ii. Analyze existing literature on using intelligent mobile agents in cognitive radio networks
iii. Define a framework for spectrum resource management and distribution, based on intelligent mobile agents.
iv. Develop intelligent mobile agent codes that can be injected into the cognitive radio network to enhance spectrum detection, decision, spectrum mobility and spectrum sharing.
v. As a proof of concept, justify how using intelligent mobile agents in cognitive radio networks leads to a better and efficient spectrum usage and distribution amongst cognitive radios.

1.7 Motivation

Efficient utilization of the radio frequency spectrum is the main idea behind cognitive radio technology. But individual radios lack the ability to reliably detect the state of the spectrum, to decide if it is free or not and at the same time transmit. This may cause interference to the primary user. A method needs to be devised to ensure that the sensing result is correct.

Furthermore, the available spectrum needs to be distributed or shared amongst the radios in a fair and organized manner so that each radio in the network will not suffer from negligence or starvation which may lead to poor quality of service on the part of the neglected radio.

A solution that addresses this problem is an intelligent mobile agent in the network, which can detect, decide and distribute spectrum resources amongst the radios in the network.

1.8 Research Methodology

The research methods to be used in this project work are as follows:

1. **Literature Survey:** this method entails surveying the background of the area of interest and analyzing related works. Previous and existing applications of intelligent mobile agents were closely scrutinized. A further study was carried out on the existing methods which led to the approach proposed in this research.
2. **Framework Formulation:** the theoretical knowledge gained from the survey will be used as a foundation for this research and to formulate our proposed framework.
3. **Code Design:** Write JAVA codes for the mobile agents that will be injected into the network.
4. **Results Evaluation:** the IMACRN will be implemented in Java Agent Development Environment (JADE) as proof of concept and a simulation of the implementation will be developed to evaluate the results.

1.9 Key Terminologies

Cognitive Radio: Radios that can dynamically sense and adapt to their operating environment. They are able to change their transmission parameters based on changes in the spectral environment.

Dynamic Spectrum Access: A method whereby, unlicensed users can opportunistically make use of licensed frequency spectrum, when the licensed users are not using them.

Primary User: A user that has the license to operate on a certain frequency band.

Secondary or Cognitive User: A user that does not have the license to operate in a particular frequency band, but can opportunistically utilize the band when the licensed user is absent.

Software-Defined Radio: Is a radio whose channel modulation waveforms are defined in software and software defined wireless communication protocols. It is capable of being re-programmed or reconfigured to operate with different waveforms and protocols through dynamic loading of new waveforms and protocols.

Agents: Agents are programs that perform certain or specified tasks on behalf of the user.

Mobile Agents: agents that perform a user's task by migrating and executing on several hosts connected to the network.

Intelligent Mobile Agents (IMA): agents that perform not only actions pre-specified by a user but also those necessitated by later changes in the environment.

Agent Migration: is a mobile action to transfer an agent with its run-time state from one host to another, without degrading the real-time performance of the cognitive radio network application.

Agent Cloning: agent remote cloning is a mobile action to create a new instance of an agent with its run-time state at another host.

Java programming language: is a concurrent, type-safe, class-based, and object-oriented language.

Java Agent Development Environment (JADE): is a software framework to make easy the development of multi-agent applications

1.10 Research Contribution

The main contribution of this research to academia and research community is the development of intelligent mobile agent codes and a generic framework for efficient spectrum resource management in a cognitive radio network.

1.11 Included Publication

Part of the research reported in this thesis has been accepted for publication. The paper is:

O.O. Ekabua and N. C. Eric-Nwonye. "Distributed Collaboration for Effective Cognitive Radio Networks Implementation", *in proc. of the International Conference on Wireless Networks (ICWN)*, WORLDCOMP'2012, Las Vegas, Nevada, July 16-19, 2012, pp. 356-363.

1.12 Dissertation Summary

The remainder of this research project is organized as follows:

Chapter 2 is on review of related literature and looks at what has been done on using intelligent mobile agents in cognitive radio networks and networks in general.

Chapter 3 introduces and describes intelligent mobile agents (IMA) as a method of achieving efficient spectrum resource management in cognitive radio networks. It also focuses on developing algorithms and JAVA mobile codes that will be injected into the network.

Chapter 4 presents the results obtained from agent based resource management.

Chapter 5 is the concluding chapter of this research report. A summary emphasizing this project's contributions is presented followed by recommendations. Suggestions for further work are pointed out.

Chapter 2

Literature Review

2.1 Overview

This chapter reviews related work done on using agents, mobile agents and intelligent mobile agents in cognitive radio networks and networks generally. The Agent Communication Languages (ACL) used in this research are also included and explained in this chapter.

2.2 Agents

A software agent is a computational entity, which acts on behalf of others. It is autonomous, proactive, and reactive, and exhibits capabilities to learn, cooperate, and move [6,17]. A mobile agent is a software agent that can move between locations. This definition implies that a mobile agent is also characterized by a basic agent model. In addition to the basic model, any software agent defines a life-cycle model, a computational model, a security model, and a communication model. A mobile agent is additionally characterized by a navigation model [19]. Mobile agents can be implemented using one of two fundamental technologies: mobile code or remote objects. The size of mobile agents depends on what they do. In swarm intelligence, the agents are very small. On the other hand, configuration or diagnostic agents might get quite big, because they need to encode complex algorithms or reasoning engines. However, agents can extend their capabilities on-the-fly and on-site by downloading the required code off the network. They can carry only the minimum functionality, which can grow depending on the local environment and needs. This capability is facilitated by code mobility [6].

2.3 Advantages of Mobile Agents

The use of mobile agents may have advantages over other implementations of agents. This does not imply that other technologies (like remote objects) cannot be used instead, because virtually any task that can be performed with mobile agents can also be performed with

stationary objects. However, the traditional solutions might be less efficient, difficult to deploy, or awkward. Some of the advantages of using mobile agents are:

(a) Efficiency savings: A mobile agent executes only on one node at a time. Therefore, CPU consumption is limited. Other nodes do not run an agent until needed.

(b) Space savings: Resource consumption is limited, because a mobile agent resides only on one node at a time. In contrast, static multiple servers require duplication of functionality at every location. Mobile agents carry the functionality with them, so it does not have to be duplicated. Remote objects provide similar benefits, but the costs of the middleware might be high.

(c) Reduction in network traffic: Code is very often smaller than the data that it processes, so the transfer of mobile agents to the sources of data creates less traffic than transferring the data.

(d) Asynchronous autonomous interaction: Mobile agents can be delegated to perform certain tasks even if the delegating entity does not remain active.

(e) Interaction with real-time systems: Installing a mobile agent close to a real-time system may prevent delays caused by network congestion. In Network Management systems, Network Management agents usually reside close to the hardware, so this advantage might not be obvious.

(f) Robustness and fault tolerance: If a distributed system starts to malfunction, then mobile agents can be used to increase availability of certain services in the concerned areas. For example, the density of fault detecting or repairing agents can be increased. Some kind of meta-level management of agents is required to ensure that the agent based system fulfills its purpose.

(g) Support for heterogeneous environments: Mobile agents are separated from the hosts by the mobility framework. If the framework is in place, agents can target any system. The

costs of running a Java Virtual Machine (JVM) on a device are decreasing. Java chips will probably dominate in the future, but the underlying technology is also evolving in the direction of ever-smaller footprints.

(h) Online extensibility of services: Mobile agents can be used to extend capabilities of applications, for example, providing services. This allows for building systems that are extremely flexible.

(i) Convenient development paradigm: Creating distributed systems based on mobile agents is relatively easy. The difficult part is the mobility framework, but when it is in place, then creating applications is facilitated. High-level, rapid application development (RAD) environments for agents will be needed when the field matures. It is quite probable that the flourishing tools for object-oriented programming will evolve into agent-oriented development environments, which will include some functionality to facilitate agent mobility.

(j) Easy Software Upgrades: A mobile agent can be exchanged virtually at will. In contrast, swapping functionality of servers is complicated; especially, if we want to maintain the appropriate level of quality of service (QoS) [6].

2.4 Foundation for Intelligent Physical Agents (FIPA)

The Foundation for Intelligent Physical Agents (FIPA) [44] is an international non-profit association of companies and organisations sharing the effort to produce specifications for generic agent technologies. It is an organization for developing standards in multi-agent systems and was officially accepted by the IEEE at its eleventh standards committee in 2005. Its goal in creating agent standards is to promote interoperable agent applications and agent systems. FIPA does not just promote a technology for a single application domain but a set of general technologies for different application areas that developers can integrate to make complex systems with a high degree of interoperability.

The first output documents of FIPA, named FIPA97 specifications, state the normative rules that allow a society of agents to exist, operate and be managed. First of all they describe the reference model of an agent platform: they identify the roles of some key agents necessary for managing the platform, and describe the agent management content language and ontology. Three mandatory roles were identified into an agent platform. The Agent Management System (AMS) is the agent that exerts supervisory control over access to and use of the platform; it is responsible for maintaining a directory of resident agents and for handling their life cycle. The Agent Communication Channel (ACC) provides the path for basic contact between agents inside and outside the platform. The ACC is the default communication method, which offers a reliable, orderly and accurate message routing service. FIPA97 mandates ACC support for IIOP in order to inter-operate with other compliant agent platforms. The Directory Facilitator (DF) is the agent that provides yellow page services to the agent platform [45].

The specifications also define the Agent Communication Language (ACL), used by agents to exchange messages. FIPA ACL is a language describing message encoding and semantics, but it does not mandate specific mechanisms for message transportation. Since different agents might run on different platforms on different networks, messages are encoded in a textual form, assuming that agents are able to transmit 7-bit data. ACL syntax is close to the widely used communication language KQML. However, there are fundamental differences between KQML and ACL, the most evident being the existence of a formal semantics for FIPA ACL, which should eliminate any ambiguity and confusion from the usage of the language [44,45].

2.5 FIPA ACL

FIPA uses a Set of Performatives as the Agent Communication Language (ACL). Different performatives are used for different actions as shown in Table 1 and explained [43,44,45]:

Table 1: FIPA Performatives

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

i) Performatives for requesting information

subscribe sender asks to be notified when statement changes

query-if direct query for the truth of a statement

query-ref direct query for the value of an expression

(ii) Performatives for Passing Information

inform together with **request** is the most important performative; basic mechanism for communicating information; sender wants recipient to believe info; sender believes info itself

inform-ref informs other agent about value of expression (in its content parameter); typically content of **request** message (thus asking the receiver to give me value of expression)

confirm confirm truth of content (recipient was unsure)

disconfirm confirm falsity of content (recipient was unsure)

(iii) Performatives for negotiation

cfp call for proposals; initiates negotiation between agents; content-parameter contains action (desired to be done by some other agent) (e.g.:„sell me car“) and condition (e.g.:„price < 1000$“)

propose make proposal

accept-proposal sender accepts proposal made by other agent

reject-proposal sender does not accept proposal

(iv) Performatives for performing actions

request issue request for an action

request-when issue request to do action if and when a statement is true

request-whenever issue request to do action if and whenever a statement is true

agree sender agrees to carry out requested action

cancel follows request; indicates intention behind request is not valid any more

refuse reject request

2.6 Java Agent Development Environment (JADE)

JADE (Java Agent Development Environment) is a software framework to make easy the development of multi-agent applications in compliance with the FIPA specifications. It can then be considered a middle-ware that implements an efficient agent platform and supports the development of multi agent systems. JADE agent platform tries to keep high the performance of a distributed agent system implemented with the Java language. JADE uses an agent model and Java implementation that allow good runtime efficiency, software reuse, agent mobility and the realization of different agent architectures [43].

The goal of JADE is to simplify development while ensuring standard compliance through a comprehensive set of system services and agents. To achieve such a goal, JADE offers the following list of features to the agent programmer [43]:

- FIPA-compliant Agent Platform, which includes the AMS (Agent Management System), the default DF (Directory Facilitator), and the ACC (Agent Communication Channel). All these three agents are automatically activated at the agent platform start-up.
- Distributed agent platform. The agent platform can be split on several hosts. Only one Java application, and therefore only one Java Virtual Machine, is executed on each host. Agents are implemented as one Java thread and Java events are used for effective and lightweight communication between agents on the same host. Parallel tasks can be still executed by one agent, and JADE schedules these tasks in a cooperative way.
- A number of FIPA-compliant additional DFs (Directory Facilitator) can be started at run time in order to build multi-domain environments, where a domain is a logical set of agents, whose services are advertised through a common facilitator.
- Java API to send/receive messages to/from other agents; ACL messages are represented as ordinary Java objects.
- FIPA97-compliant IIOP protocol to connect different agent platforms.
- Lightweight transport of ACL messages inside the same agent platform, as messages are transferred encoded as Java objects, rather than strings, in order to avoid marshalling and unmarshalling procedures.
- Library of FIPA interaction protocols ready to be used.
- Support for agent mobility within a JADE agent platform.
- Library to manage user-defined ontologies and content languages.
- Graphical user interface to manage several agents and agent platforms from the same agent. The activity of each platform can be monitored and logged. All life cycle operations on agents (creating a new agent, suspending or terminating an existing agent, etc.) can be performed through this administrative GUI.

- The JADE system can be described from two different points of view. On the one hand, JADE is a runtime system for FIPA-compliant Multi Agent Systems, supporting application agents whenever they need to exploit some feature covered by the FIPA standard specification (message passing, agent life-cycle management, etc.). On the other hand, JADE is a Java framework for developing FIPA-compliant agent applications, making FIPA standard assets available to the programmer through object oriented abstractions [43,45].

2.7 JAVA Programming Language

The Java programming language is a concurrent, type-safe, class-based, and object-oriented language. It was born out of a project called Oak led by James Gosling at Sun Microsystems during the early 1990s. Originally intended as a programming language for set-top boxes, Java quickly became a general-purpose language that is used in a variety of applications. Java was chosen as a base, because it is representative of a family of programming languages whose goal it is to make programming less error prone and to make programs more robust.

2.7.1 Language Overview

Java's syntax and procedural style were taken from C/C++. It uses the same control structures, same expressions, and similar primitive types and operators. However, Java is stricter in defining the behavior of types and operators. For example, unlike C, Java specifies the signedness and range of every primitive data type to ensure that a Java program has the same meaning on different platforms [10].

Objects: The basic unit of code is a Java class, which can contain fields and methods. As in C++, fields are either per-instance or (global) static fields. Java supports subclassing through single class inheritance. Polymorphism is provided through the use of virtual methods. Unlike C++, Java supports neither nonvirtual, nonprivate methods nor code outside of classes, which forces a more strict object-oriented style of programming. In addition, Java supports subtyping through interfaces. An interface is a collection of method declarations

that an implementing class must provide. A class can inherit from only one base class, but it can implement an arbitrary number of interfaces. By disallowing fields in interfaces, Java avoids the difficulties associated with multiple inheritance.

***Packages*:** Java classes are grouped in packages, which are roughly comparable to C++ namespaces. Packages are named in a hierarchical fashion. For instance, all classes that are part of the runtime library are found in the java.* hierarchy, such as java.lang.String for basic functionality such as Strings or java.net.* for the set of classes that provide access to standard network facilities such as sockets.

***Type safety*:** Java is a type-safe language; Java programs cannot arbitrarily access the host computer's memory. All memory is tagged with its type, each of which has a well-defined representation. Java does not use pointers. Instead, objects are accessed using *references*, which cannot be cast into or from any primitive type, such as an integer. Arrays are first-class objects, and all accesses are bounds-checked [10,12].

These properties avoid the pitfalls associated with pointer arithmetic. Java's memory management is automatic. The programmer is relieved from the burden of having to explicitly manage memory. Instead, a garbage collector determines which objects the program is able to access in the future; it reclaims those objects that cannot be accessed on any legal execution path. Consequently, there are no crashes due to dangling references that point to already freed memory. Note, however, that memory leaks can occur if references to objects remain stored unintentionally. Java allocates all objects on the heap—there are no locally declared static objects as in C++. This design avoids the complications associated with copy constructors and destructors in C++. Instead of destructors, Java supports finalization of objects. Objects can provide a finalize() method that is executed once the object has become unreachable but before its memory is reclaimed. Finalize methods can be used to free system resources, such as sockets, that may be associated with a Java object.

***Language access modifiers*:** Java supports information hiding using access modifiers. Each class, method, or field can be marked with either *private*, *protected*, *public* or can have no

modifying prefix at all. Private asserts that a field or method can be accessed only from within its class. Protected fields and methods are accessible to subclasses, and public fields and methods are accessible from anywhere. In the default case, a field or method can be accessed only from classes within the same package as its class [10,13].

Access modifiers are enforced at runtime. Consequently, access to critical fields can be effectively protected by declaring them private. The flexibility of language access modifiers is limited, because there is no feature similar to C++'s friends that allows a class to grant access to a specific class or method. Therefore, accesses from outside the current package require that the data or method be public, which makes it accessible to all packages. Run-time checks are then required to determine whether the caller has sufficient privileges to perform the requested operation.

Exception handling: Java exceptions are used to signal errors that occur during program execution. Java adopted C++'s try/catch model for exceptions. If an exception is raised within a try block, execution branches abruptly from the current instruction to the corresponding catch clause for that exception. If there is no matching clause, the method terminates abruptly, and the exception is thrown in the caller. In this way, exceptions propagate up the call stack until either a matching catch clause is found or there are no more stack frames. Unhandled exceptions terminate the current thread. An exception can be raised in different ways: either directly by executing a throw statement or indirectly as a side effect of executing a bytecode instruction or runtime operation. For instance, any array store instruction can throw an exception if the array index is out of bounds. Similarly, attempts to dereference a null pointer results in a null pointer exception. Some runtime operations can be interrupted on request; such interruption is also signaled via an exception [13,14].

There are two groups of exceptions: checked exceptions, which an application is expected to handle if they occur, and unchecked exceptions, from which an application is not expected to recover. Every method must declare the checked exceptions it can throw. The compiler checks that any caller of a method that can throw an exception is prepared to handle it—either by supplying a compatible catch clause or by declaring the exception itself. The

rationale behind this rule is to avoid a common pitfall in languages such as C, where programmers tend to forget to check return codes for error conditions [10,12].

Unchecked exceptions need not be declared. Java's designers judged that having to declare such exceptions would not significantly aid in establishing the correctness of the code and would pointlessly clutter code [10]. In addition, it would be difficult or impossible for a program to recover from them. An example of such an exception that is hard to recover from would be a loading error that occurs if parts of the program code cannot be loaded. Note that code in critical parts of the runtime system must handle all errors, including those errors that signal resource exhaustion, such as out-of-memory errors. Consequently, such code may not be able to take significant advantage of Java's exception checking rules.

Concurrency: Java is a concurrent language with built-in support for multithreading. Programs can create new threads simply by instantiating an object of type java.lang.Thread. Threads have their own stacks for their local variables, and share data through objects on a common heap. Two kinds of synchronization are supported: mutual exclusion and unilateral synchronization. Mutual exclusion is provided through the synchronized keyword, which encloses blocks of code in a critical section that can be entered only by one thread. In addition, methods can be declared synchronized, which is equivalent to enclosing the entire method body in a synchronized block. Java's concurrency mechanisms do not prevent unsynchronized accesses to shared variables [11]; consequently, race conditions can occur. Unilateral synchronization is provided in a wait/notify style [12]; every object inherits a wait() and a notify() method, and therefore every object can be used as a condition variable.

Every object to which a program holds a reference can be used as a lock when it occurs as an argument to a synchronized clause. This feature can, however, cause problems when public fields of critical objects are used for synchronization. For instance, in an early version of the HotJava browser, an applet could effectively halt all activity in the browser by acquiring and not releasing a lock on its status bar object. Other applets and the system would get stuck as soon as they tried to display any messages in the browser's status bar.

Terminating uncooperative threads is not safe in current versions of Java. To understand this problem, it is useful to look at the history of Java threads. The original Java specification provided a method called Thread.stop() to stop threads. This method caused an exception to be thrown in the targeted thread's context. This asynchronous exception was handled like any other run-time error, in that locks were released while unwinding the stack. Later, JavaSoft realized that this procedure could lead to damaged data structures when the target thread holds locks.

In particular, data structures vital to the integrity of the runtime system could be left in inconsistent states. As a result, JavaSoft deprecated Thread.stop(). A later proposal for termination was also flawed. A different mechanism for termination, Thread.destroy(), was proposed by Sun that would have terminated a thread without releasing any locks it held. This proposal had the potential for deadlock, and JavaSoft never implemented it. There is currently no supported mechanism to ensure atomicity in the presence of asynchronous termination [11,12,14].

***Native libraries*:** In many circumstances, it is necessary to interface Java code with code in other languages, usually C or assembly code. To access such code, Java allows methods to be declared native, i.e., as having an implementation in non-Java code. Since C code cannot be shown to maintain Java's safety properties, only trusted classes can have native code, although some systems have tried to avoid that restriction by placing the code in a different operating system process [13]. However, such approaches require each native call to use interprocess communication mechanisms.

Writing native code requires close cooperation with the JVM's garbage collector. The code must ensure that objects that are kept alive by native code are found by the collector. Two approaches are possible: a conservative collector can scan the data segment of the entire program and keep all objects that could be referenced alive, or the code is required to explicitly register and unregister references it needs to be kept alive. The latter approach is chosen in the Java native interface (JNI) specified by Sun Microsystems [14]. Native code is allowed access to Java objects only through a well-defined interface, which includes such

operations as getting or setting fields in an object. This interface allows the JVM to closely monitor all operations done by native code that are relevant to the garbage collector.

2.8 The Java Virtual Machine

A Java compiler translates Java programs into bytecode that is targeted at the Java virtual machine. Java bytecode provides the basis for Java's portability, because the same bytecode can be executed on all platforms that provide an implementation of the Java virtual machine. Java uses late binding (Java bytecode is loaded and linked at run time). Code can be loaded from a filesystem or from user-specified sources such as web servers. After the bytecode is loaded, it is checked by a verifier before being linked. If the verification is successful, the bytecode is executed. Java bytecode can be executed by an interpreter, or it can be translated into native code by a just-in-time compiler (JIT) first. Most JVMs today employ JIT compilers; interpreters are typically used only where the space and complexity overhead of JIT compilation is not tolerable, such as in small or embedded systems. To avoid the overhead of JIT compilation for code that is executed only rarely, some VMs use an adaptive technique that interprets code first and translates those pieces that are invoked frequently during the execution of the program.

The Java virtual machine is a simple, stack-based virtual architecture that provides storage for the heap and each thread's stack and registers, and a CPU to execute bytecode instructions. Local registers, object fields, and static fields can be pushed on and popped off the stack. All arithmetic and logical operations are performed on stack elements; their results are also stored on the stack [14,15,16].

***Bytecode verification*:** Bytecode verification ensures that the bytecode maintains the same safety properties that are present in the source language. Java bytecode is designed such that the Java runtime can verify its consistency and therefore need not trust the bytecode compiler to produce proper code. This property allows the use of Java bytecode in mobile code environments. A client can check safety properties of the code without access to the source

code. Bytecode instructions are strongly typed, so that the bytecode verifier is able to verify whether the type of the operands matches the expected type.

***Class loading*:** Java's class loading mechanism provides four major features: lazy loading, multiple namespaces, user-defined loading policies, and type-safe linkage [15]. Lazy loading postpones class loading and linking until a class's first use; users can define class loaders to provide their own loading policies and define multiple namespaces. The JVM ensures that the linking process cannot be used to bypass Java's type-safety guarantees.

A *class loader* is an object that serves requests for class objects. A class object is a runtime representation of a type. Each class object is tagged with the loader that loaded it. The VM maintains a mapping:

(class loader, class name) ⟶ *class object*

All class objects that have a loader in common form a namespace, for which the common loader acts as name server. Class objects with the same name that are loaded by different class loaders constitute different types: attempts to cast one to the other fail. Multiple namespaces can be created by instantiating multiple class loaders. Supporting multiple name spaces is necessary for applications such as running applets from multiple sources in a browser, because there is no prior coordination between the different developers as to how to name their classes. Class loaders *define* class objects by presenting a sequence of bytes in class file format to the JVM. The JVM parses and verifies the contained bytecode, establishes a new runtime type, and returns a reference to the new class object. The loader that defines a class is said to be that class's defining loader. Class files contain both a class's bytecode and symbolic information necessary to resolve interclass references [14,15,16].

If a class contains symbolic references to another class, the runtime system initiates the loading process of the referred class by requesting a class object from the referring class's defining loader. The initiating loader can either load the necessary class file itself, or it can in turn invoke another class loader to resolve the reference. This process is known as *delegation* [15]. Since the Java type is determined by the defining loader, delegation allows different class loaders to share types by delegating them to a common loader. One example is the

system class loader, to which all loaders refer for system classes. Because a class loader is a user-defined object, it has full control over the location from where class files are loaded. Ordinarily, class files are loaded from a local file system. A class loader can implement other options: for instance, a loader in a mobile agent system could load class files over the network from an agent's originating host. Some systems even create class files on the fly from other source languages, such as Scheme. Class loaders can also restrict their classes' access to other classes by refusing to return an answer for a requested name. This mechanism can be used to enforce a crude form of access control.

***Type-safe linkage*:** The virtual machine must guarantee type safety, even in the presence of class loaders that do not return consistent answers when asked for names. For instance, a class loader should return the same runtime class when queried for the same name. The JVM must be able to cope with user-defined loaders that violate this invariant. An insufficient understanding of the consistency assumptions made by the JVM compromised type safety in early JVM implementations. Ill-behaved class loaders were able to bypass language access modifiers or could even cause the JVM to crash. These vulnerabilities were fixed by defining and enforcing a set of constraint rules on the classes a loader defines [15]. For instance, one rule states that if a class references a field declared in a class that was defined by a different loader, then that field's type must have the same definition in both loaders' namespaces. If a loader attempts to define a class that would violate such constraints, a linkage error is raised.

***Security model*:** Java's security mechanisms defend against threats that can arise from untrusted code, such as the possibility of system modification or corruption or leakage of sensitive information that could violate a user's privacy. Untrusted code is prevented from performing operations such as writing to or deleting local files or directories, or reading from files and sending their contents over the network. The Java security model associates principals with code and controls access to resources such as files and sockets. Code that originates from the same source is considered to belong to the same principal and is given equal privileges [13,15].

Java's designers have tried to separate security mechanisms and policies by centralizing policy decisions in one location in the runtime system. Early versions of Java used a SecurityManager object for that purpose. Runtime classes consulted the security manager whenever an access control decision was to be made. If the attempted operation was disallowed, the security manager vetoed it by raising a SecurityException. The first policy implemented using security managers was the *sandbox* policy. The sandbox policy is an all-or-nothing approach: code from untrusted sources is sandboxed and has no privileges, and trusted code has all privileges. The security manager bases its decision on whether to allow a sensitive operation to proceed on the call stack of the current thread: if any activation record belongs to untrusted code, the operation is disallowed. Because this approach was too inflexible, later versions of Java replaced it with an approach that does not require a security manager but supports making access control decisions directly in the VM. Classes are placed in different *protection domains*, and each domain possesses a set of permissions. When an access control decision must be made, the JVM examines the call stack and determines the protection domains in the current thread's call chain [14,15,16].

Unlike in the security manager model, the JVM obtains the effective set of permissions by intersecting the permissions held by the domains that are part of the call stack. This algorithm implements the principle of least privilege: it prevents lower-privileged code from acquiring privileges by calling into higher-privileged code, and it requires higher-privileged code to effectively give up its permissions when calling into lower-privileged code [15,16].

2.9 Intelligent Mobile Agents in Cognitive Radio Networks

Some researchers have applied multi-agent systems for dynamic spectrum allocation and sharing in the context of cognitive radio networks. A multi-agent system based solution has been proposed to achieve licensed and unlicensed dynamic spectrum sharing. Firstly, the authors [17] presented a cooperative approach where the cognitive radio nodes embarked with agents are capable of performing spectrum sharing by exchanging a series of messages

with the neighboring licensed devices. While analyzing the performance of this proposal under ad-hoc wireless conditions, the authors show that it achieves good performance in terms of spectrum access, without incurring greater communication cost. Then, they focus on enabling unlicensed spectrum sharing between the cognitive radio users. Their proposed solutions can achieve good performance while maintaining fair spectrum distribution [17].

Another multi-agent systems based spectrum sharing strategy considers cooperative multi-agent systems, in which the agents are deployed over primary and secondary user devices. The developed cooperation mechanism is similar to that of contract net protocol (CNP), in which the individual secondary user (SU) agent should send messages to the appropriate neighboring primary user (PU) agents whenever needed and, subsequently, the related primary user agents should reply to these agents in order to make a spectrum sharing agreement. The paper [18] proposes that the secondary user agents should make their decisions based on the amount of spectrum, time and price proposed by the primary user agents and should start spectrum sharing whenever they find an appropriate offer (without waiting until the reception of all the neighboring primary user agents' responses). Then, after completely utilizing the desired spectrum, secondary user agents should pay the agreed price to the respected primary user agents [18].

Cooperative radio resource management for multiple cognitive radio networks in interference environments has been investigated by [19] and the objective of the research was to manage shared radio resources fairly among multiple non-cooperative cognitive radio networks to optimize the overall performance. A multi-agent system-based approach was then proposed to achieve information sharing and decision distribution among multiple cognitive radio networks in a distributed manner [19].

A multi-agent learning algorithm that is applied for optimizing online resource allocation in cluster networks was proposed by [20], in which the learning is distributed to each cluster, using local information only and without access to the global system reward [20]. For efficient radio resource utilization, a novel intelligent multi-agent radio resource management system is proposed, which is self-organized and distributed to ensure the

coexistence of multi-radio access technologies. Radio resource is managed by a macro control and management system using control factors and validation mechanism, instead of micro control for individual users. The goal is to increase radio resource utilization efficiency, maximize system capacity and meet the quality of service (QoS) requirements of different services [21].

Finally, work was done on using multi-agent systems that adopt mobile agents as a technology for tasks distribution, results collection, and management of resources in large scale distributed systems. A new mobile agent-based approach for collecting results from distributed system elements was presented and the technique is based on intelligent agents giving the system a proactive behavior. As mentioned above, we can use multi-agent systems for dynamic spectrum allocation and sharing in the context of cognitive radio networks [22].

2.10 Cognitive Radio Resource Management Using Multi-Agent Systems

Cognitive radios provide a potential solution for more efficient spectrum utilization. To achieve efficient spectrum utilization, a balanced and integrated communication system is required [26,27]. One solution is to incorporate spectrum management functionality with the software-defined radios attributes in communication systems. Cognitive radio resource management requires a tight coupling between the spectrum management functionality and the software-defined radios attributes, i.e., modes of operation supported by the physical layer. Wireless local area networks (WLANs) provide essential components for projected cognitive radio platforms. Since predictive models can be readily developed for current WLANs, they make an ideal hardware platform for developing our resource management strategy.

2.10.1 Architecture of Cognitive Radio Resource Management Using Multi-Agent Systems

One application of cognitive radio resource management is the multi-domain WLAN environment. In recent years, many hot-spots are emerging and multiple WLANs are being deployed within small geographic vicinities. Different WLANs in a particular area may be deployed by different operators. In such a multi-domain environment, there is a growing interest in WLAN providers setting up reciprocal agreements so that mobile users may share the usage of multiple WLANs. A multi-agent system-based approach is proposed to achieve information sharing and decision distribution among multiple WLANs in a distributed manner. WLAN providers may set up service-level agreements among themselves on how much data can be exchanged among agents. Compared to using a centralized controller, a multi-agent system-based approach is more scalable.

A resource management architecture for multiple WLANs using multi-agent systems was proposed by [6], whereby multiple WLANs are co-located within a particular geographic area. Communications inside the surrounding wireless personal area networks (WPANs) such as Bluetooth networks and wireless sensor networks (WSNs) generate interference to WLAN activities. Agents are located inside each access point (AP) and interact with other agents within its neighborhood. An agent's neighborhood consists of those agents with whom it has frequent interactions. These interactions include sharing of data and negotiating about resource assignments. Individual agents act as radio resource coordinators and cooperate with agents in their neighborhood to take care of resource management across multiple WLANs.

Through agent coordination, providers may offer inter-WLAN roaming services to their subscribers as a value-added service feature. They can also support communications with better quality signals since the impact of interactive interference can be globally balanced through multi-agent control. The functions related to user authentication, billing, security and privacy, and mobility management can also be implemented in agents. Within the multi-agent system, the agents are leveraged to fairly balance system-wide resources in order to

accommodate more users with the least amount of cost. The agent at each AP collects the statistics from the measured operational environment as well as its neighborhood and estimates the required parameters for optimizing system performance based on predictive models. The agents use the measured data to generate local control decisions and try to optimize the performance of the entire WLAN system in a distributed fashion through agent interaction and coordination. Agent interaction is an essential aspect of this architecture. Agent interaction occurs on the backbone network connecting all the APs. Therefore, the bandwidth requirement for agent interaction is not a critical issue. However, since multiple agents contribute to the control of optimal resource allocation across WLANs, they need to decide what information should be exchanged among neighbors, how often to exchange this information, and which neighbors should act as relay nodes for the data. When a control decision is made, an agent also needs to decide what actions its effector should take and how the control decision should be distributed to the desired area [6].

The major functional blocks of a general framework for physical environment prediction and resource management using agent technologies are: WLAN and WPAN cluster, RF environment sensing (RES), and agent operations which include predictive parameter estimation (PPE) and resource management optimization.
They are explained in details as follows.

i) WLAN and WPAN Cluster: Each mobile station (MS) in WLANs operates within a dynamic RF environment comprising time-varying co-channel interference sources and time-varying interference sources from co-located WPANs. The agents inside each AP periodically collect measured statistics from the dynamic RF environment required for resource management.

ii) RF Environment Sensing (RES): This block is used to provide estimates of the signal characteristics from both MSs within the WLAN cluster as well as potential interference sources within the operational environment. Part of the functions defined in this block can be provided by the specifications of IEEE 802.11k radio resource measurement. Statistics related to WPAN environmental interference levels should be provided from an additional

sensing component inside each AP. It is important to remark that it does not imply measuring instantaneous small-scale multipath signal characteristics which are very time-sensitive. Instead, measurements would be targeted at capturing large-scale changes in signal characteristics due to variations in shadowing, MS mobility, interference sources, and interference locations. In other words, the RES needs to measure the factors which influence the resource management of the WLAN performance.

iii) Agent Operation-Predictive Models for Parameter Estimation (PPE): Estimates of signal characteristics are input to the agent inside each AP. An agent also receives data from its neighborhood through agent interaction and coordination. The general concept for the PPE block is to use predictive models to generate parameter estimates required by the resource management optimization. The parameters to be estimated include:

- **Link Quality:** link quality between each MS and its AP.
- **Mobility Rate:** rate of changes in the expected link quality between each MS and its AP.
- **Energy Expenditure:** energy required to successfully transmit a packet between each MS and its AP.
- **Throughput:** throughput for each WLAN cell based on the operational environment characteristics, current offered traffic, and projected offered traffic.
- **Latency:** expected time delay and the variance in the time delay between each MS and its AP.

iv) Agent Operation-Resource Management Optimization: This block analyzes the parameter estimations and makes instructional decisions to optimize the overall WLAN performance based on designed optimization models. Instructional decisions include the optimal transmit power at APs, the optimal channel APs should operate in order to minimize interference levels and make the best use of overall resources, whether or not to accept association requests from specific MSs, whether to direct specific MSs to be associated to another AP for load balancing, and so on. These decisions are updated periodically in order to address changes in the traffic load and interference environment. They should target long-

term performance improvement. The operational changes are downloaded to the WLAN cluster with the help of agent effectors and distributed to the neighborhood of agents through agent interaction and coordination.

The resource management optimization block includes two components:

i) Utilization Modeling and Optimization (UMO): This block finds the optimal utilization, i.e., the maximum allowable throughput, of each AP based on the environmental information agents possess. The decision of the optimal utilization is used by the Effect Optimal Utilization (EOU) block (which is explained in the following), to generate specific strategies to achieve the optimal utilization at each AP.

ii) Strategy to Effect Optimal Utilization (EOU): Given the optimal utilization of each AP, instructional decisions are generated to achieve the optimal utilization while minimizing interference to the environment. Operational changes are negotiated within the agent's neighborhood and applied to the WLAN cluster. They are also fed back to the UMO block to update the optimal utilization decision [6,28].

2.10.2 Third-Party-Based Architecture

A third-party-based resource management architecture was proposed to facilitate the cooperative multi-domain resource management. A trusted third-party agent is needed who is independent from each network provider's financial interests. When a new WLAN is deployed, the WLAN provider does not need to set up direct service level agreements with all the other providers of the existing WLANs in the area. It only registers to the third-party agent. The third-party controller can collect information across multiple domains and send control signals back to each domain, thereby making radio resource management and other features possible [1]. A new entity, Local Network Controller (LNC), is connected to all the APs of multiple WLANs. WLANs under the control of an LNC form a WLAN cluster. The LNC acts as a radio resource coordinator across domains and takes care of issues related to inter-domain roaming and resource sharing within a WLAN cluster. As the number of

domains in a WLAN cluster increases, the LNC can be built in a hierarchical structure to make it more scalable. A global network controller (GNC) is connected to all LNCs supporting inter-WLAN-cluster roaming and resource sharing [6,29].

The LNC gathers the measured resource usage statistics from all the APs via Simple Network Management Protocol (SNMP) [29]. APs collect signal characteristics from client stations in each domain based on IEEE 802.11k specifications [28]. The measured data can then be used by the LNC to generate control decisions to optimize the performance of the entire WLAN cluster. The goal of the proposed scheme is to minimize the total system cost by adjusting resource allocation in each domain. The cost is what the system needs to pay to support all the MSs to achieve performance requirements. It is related to the available radio resources for supporting the offered load in each domain and mitigating interference from the operational environment. The LNC manages resource sharing across domains by controlling the maximum allowable throughput of each AP. When the maximum allowable throughput at an AP changes, the available radio resources of the cell is limited. Consequently, the cell utilization changes which leads to a different system cost. Therefore, minimizing the overall system cost is equivalent to finding the optimal allowable throughput at each AP. In addition, WPAN interference can adversely affect the WLAN performance by changing its resource utilization requirements and therefore needs to be considered. Moreover, due to the dynamics in the RF environment, signal characteristics, traffic load, and interference intensity are time-variant. As a result, the optimal resource allocation decision should be dynamically adjusted to reflect the influences of the time-varying environment.

The proposed resource management scheme [6] includes three steps. First, based on the overall traffic load distribution at all the APs in a WLAN cluster, the impact of co-channel interference at each cell can be calculated. Then, by incorporating the impact of interference from other sources in the operational environment, the communication cost of the overall system can be derived which is a function of cell load, co-channel interference, and interference from other wireless services.

Second, the LNC finds the optimal pattern of maximum allowable throughput at each AP in multiple domains. In other words, the LNC decides which AP should provide how much capacity to its users. This optimal throughput pattern results in the minimum system cost.

Finally, the LNC sends control signals to APs to instruct them on how to update their allowable resources for users based on the calculated optimal throughput. The proposed multi-domain resource management scheme is able to minimize the co-channel interference across domains and mitigate other interference from the operational environment through fair resource allocation.

Under the proposed scheme [6], resource utilization and co-channel interference can be adaptively balanced across the entire integrated system. Simulation results demonstrated that the proposed multidomain cooperative resource management scheme is more cost-efficient for a WLAN/WPAN interference environment. The proposed scheme can save up to 99.8% and 47.3% cost compared to the scheme that each domain optimizes resource usage independently without the consideration of potential interference from co-located WPANs and the scheme that LNC is involved to help control the resource allocation in each domain but without the consideration of potential interference from co-located WPANs, respectively [6,28,29].

2.11 Multi-Access Radio Resource Management using Multi-Agent Systems

Next generation networks are characterized by the efficient integration of multiple radio access technologies in a common network, and terminals with reconfigurable capabilities, thus imposing new requirements on the radio resource management (RRM). Conventional RRM cannot cope with the increasing demands imposed by both networks and users. Multi-access Radio Resource Management (MRRM) has come into the spotlight only in the last three years. Existing research efforts are mostly focused on centralized management, which means that one or more central entities coordinate different RAT-specific RRM entities.

In [23,24], Joint Radio Resource Management (JRRM) is presented by Siemens. The core concepts of this management system are "service-split" and "multi-homing". JRRM splits the service into fundamental part and enhanced part and the former is delivered by RAT with large coverage range, e.g. UMTS, while the latter by RAT with high bandwidth, e.g. HIPERLAN/2. From the perspective of future business development, however, the centralized RRM mode is unrealistic, since a central management entity coordinating RATs is required. It should be noted that existing and emerging RATs will be operated by different network providers (NPs) in the future. Generally, NPs do not share information regarding the exact structure of their networks, especially the cell layout. Therefore, it is unnecessary and impossible to establish a central management entity to coordinate the heterogeneous networks. An alternative solution is adopting distributed Multi-access Radio Resource Management mode. In [25], the relationship of distributed networks is discussed and a comparison between centralized and distributed modes is presented.

An analysis of the state-of-the-art MRRM in heterogeneous networks reveals that almost all efforts attempt to construct the management platform or design new mechanisms to improve the network-centric performance. Most of them handle the detailed RRM decisions concerning individual users or cells, thus increasing the network loading significantly. In the meantime, an important issue, the coexistence of Multi-RATs in the same geographical area, is neglected. In the future, each RAT should possess a certain market share, since each technology is tailored to reach a particular market, or particular type of user with specific service need. Therefore, a question naturally arises: how does one guarantee the coexistence of different RATs?

To address the aforementioned unresolved problem, a novel distributed intelligent Multi-Agent System (MAS) was proposed in [8], which is self-organized to solve the problem due to the coexistence of multi RATs operated by different NPs. Compared with the issues addressed by the existing JRRM, MAS is mainly concerned with the coexistence of different RATs in a specific geographical area. The agent performs a macro control and management function by using control factors and validation mechanism instead of micro control handling detailed RRM decisions concerning individual users [8].

2.11.1 Network Architecture

The Multi-Agent System has three kinds of radio access networks. The MRRM functions are performed by the agent, which is isolated from the RAT specific RRM functions. Only the pair of agents that handle overlapping cells needs to have a peer-to-peer relation. The agent is responsible for collecting the status information of the underlying specific RAT and the information of neighboring agents. Based on the obtained information, the agent deals with the coexistence-related issues, including optimization of the Objective Function (OF) of the local network and satisfaction of the Boundary Function (BF) of the neighboring network, and inter-network handover. On the one hand, according to OF, the adjustment performed by the agent can overcome adverse situation (e.g. overload).

On the other hand, according to BF, the adjustment can guarantee the coexistence of both RATs. In other words, the adjustment will not cause destructive effects on neighboring RAT. As for deployment, taking WCDMA for example, one agent can control one Radio Network Subsystem (RNS) or several RNSs according to the specific traffic volume of the geographical area. It should be noted that the agent is a logical entity. Therefore, the agent can be a standalone server or be integrated into other existing management entities, e.g. RNC. With respect to granularity, since the user distribution and service traffic of each cell differ from others, the agent collects the status information and performs adjustment on a cell basis. Regarding cost, the agent is required only in the scenario where the two RATs overlap with each other in the same geographical area.

The relationship between agents is self-organized. Each agent has cooperative capabilities and competitive capabilities, trying to maximize the outcome of a specific objective. However, the global objective of the system is achieved through the collaboration of all agents [8].

2.11.2 The Basic Entities and Interfaces

The key element in the architecture is the agent, which is a new functional component to support multi-access radio resource management. The agent performs all the resource management and control functions for radio networks with overlapping cells. The agent is an intelligent element, which collects the information both from the radio network under its control and from other agents. It can actively search cells overlapped with other radio networks and actively detects whether it is necessary to trigger resource adjustment according to the received information. When performing resource adjustment, the agent not only considers the requiremenst of the local network, but also the requirements of networks with overlapping cells. The agent performs macro control by using control factors and validation mechanism [8,24].

The interfaces involved are briefly discussed below.

1. **Hr:** This interface is used to carry information between the RAT and the corresponding agent. Through this interface, an agent collects radio network information such as parameters related to radio resource, parameters about performance, etc., and sends control factors to the RAT.

2. **Ha:** The Agent-Agent interface is used to carry information concerning:
    a. Inter-network resource adjustment and coordination
    b. Status information of each other

2.11.3 The Relationship between Agent and other RAN Functions

Naturally, an agent is located at the control plane [25], but that is not to say that it is completely decoupled from the user plane. In fact, there are functions that can be classified as agent functions that benefit from being closely coupled or even integrated with the user plane functionality. Moreover, an important control plane role of RRM and Agent is to handle configuration and reconfiguration of the user plane.

A. Cognitive Layer

The cognitive layer monitors the controlled radio network and other agents. It can find overlapping cells according to the information either from RAT or from other agents, and can also determine when to perform the adjustment. The cognitive layer comprises the radio cognitive module (RCM) and the cooperation cognitive module (CCM). RCM collects the status information of the controlled RAT via the *Hr* interface. RCM perceives automatically whether it is necessary to trigger control or adjustment of the reaction layer. Within RCM, a set of thresholds configured by the reaction layer are defined, e.g. maximum loads, maximum delay, maximum packet loss, etc. Once any of the above statistics exceeds its corresponding threshold, the adjustment is triggered. Alternatively, the agent may execute the adjustment periodically to ensure the operation status is on track. Besides, RCM also transfers control factors and other control information to the control RAT. Some control factors are for the radio network, and others are for the end users. The trigger threshold could be adjusted as per the real situation by the control subsystem. RAT broadcasts adjustable factors and other parameters to mobile terminals, and the network is selected accordingly. Each agent has its own eigenvector that identifies the agent's features, which is composed of parameters related to resource and control factors. CCM collects the eigenvectors of other agents via the *Ha* interface and the evaluation results of other network. CCM perceives automatically whether it is necessary to trigger control or adjustment of the reaction layer.

B. Reaction layer

The control module of the reaction layer is the core control module of the agent and achieves macro-control of the heterogeneous network radio resource. The computation of the control module is triggered by either the change of local RAT or the change of neighbor agents with the overlapping cells. The control module consists of the computation module and the validation module. The functions of the control module are described as follows:

1) Setting trigger thresholds of cognitive layer: The thresholds would change with the change of time and are related to the radio transmission environment. These data are stored in the memory layer, and the setting of the trigger thresholds is performed adaptively by the reaction layer.

2) Managing overlapping cells: With the information from the cognitive layer, the control module can get the overlapping information with other RATs; then, the control module will store the information in the memory layer and notify the agents of the overlapping cells.

3) Adjusting radio resource: The objective function in the control module is used during the adjustment of the radio resource to deal with the coexistence of different RATs. In order to realize the optimization of resource in different radio networks and meet the QoS requirements of different services, the objective function is not only related to the satisfaction degree of the users in the local network, but also related to the satisfaction degree of neighbor-network covering the same area. The objective function is optimized by adjusting the control factors.

4) Using control factors: Control factor is a set of variable parameters used to direct each subscribers to select the most suitable RAN in order to adjust the distribution of services. It should contain the following parameters: price and admission control parameters in the network side such as radio network load, mobility capability supported and radio network access conditions. The control factors of the radio network can only be changed by the agent which controls this radio network.

5) Coordinating with the neighboring agents: When the adjustment of resource is triggered by other agents, the agent can evaluate whether other agents' behaviors are proper, and then respond with the evaluation result to that agent whose eigenvector parameters have been changed. If the evaluation result is positive, this agent will not be changed. On the contrary, if the evaluation result is negative, this agent will also adjust control factors and the respective eigenvector parameters. After a period of adjustment, the system will achieve a new balance.

6) Verifing the adjustment results: In the control module a validation mechanism is used to evaluate whether the adjustment is appropriate. This validation mechanism is based on interactive procedures between different RATs, and is achieved by the validation module.

C. Memory layer

To accelerate the adjustment and for intelligent control, the memory module is adopted in the memory layer. For each successful adjustment, the memory module should save the results. When the computational module faces a similar situation, the existing suitable adjustment results can be obtained directly from the memory module rather than by recalculation and re-verification. The control subsystem can also get the control policy from the memory layer. Besides, this module has self-learning capabilities, i.e. it can learn from other agents and from itself to enhance efficiency [8].

2.12 Intelligent Mobile Agents in Wireless Sensor Networks (WSN)

Wireless Sensor Networks (WSN's) have been identified as one of the most important technologies for the 21st century [30]. The deployment of such applications, usually to harsh physical environments, is essentially permanent resulting in a high probability that some nodes will fail and that experimental parameters, such as the frequency of sensing will change. A representative WSN application will be based on the TinyOS (a component-based embedded operating system) platform and a set of MICA2 (a mote module used for WSNs) motes [31]. TinyOS builds its architecture in a component based model triggered by event driven calls. Using TinyOS, it is not possible to create programs which can be dynamically modified e.g. changing an input parameter [32]. Additional complications are the limited physical memory of each mote and the extremely unreliable radio communication [33].

To address the above limitations, an intelligent mobile agent middleware for WSNs called In-Motes, was developed. In-Motes is based on Agilla (a mobile agent middleware designed

to support self-adaptive applications in WSNs) and Mate (a tiny virtual machine for sensor networks). However, unlike Mate, which is based on the generation of a virtual machine that divides each application into a specific number of capsules that are flooded throughout the network, In-Motes follows the Agilla concept that allows users to inject agents into the network [32,34,35].

In order to overcome the Agilla's memory and network management limitations, In-Motes provides a high level architecture for the agent community based on a federated system, and behavioral rules based on a biological analogy with bacterial strains [29,36]. Mobile agents can intelligently move or clone themselves to desired locations in response to changing conditions in the environment. In-Motes was implemented on the MICA2 and TinyOS platform and the design and preliminary performance micro-benchmarks of In-Motes were reported [37,38].

The In-Motes architecture is shown in Fig. 2.1, and it's divided into three layers, which are, the agent manager, the rules manager and the facilitator manager [39].

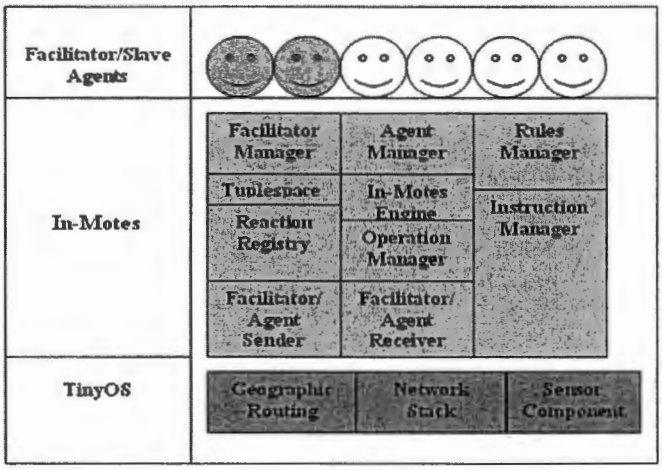


Fig.2.1. The In-Motes Architecture

The first layer consists of the agent society and the In- Motes layer sits on top of the TinyOS platform.

The Agent Manager is responsible for maintaining the context of both facilitators and slave agents. When an agent of either type arrives in a mote, the agent manager allocates to the entity a specific amount of memory. It is also responsible for notifying the In-Mote engine when an agent is ready to run in the network. Subject to the constraints imposed by the available hardware, the agent manager is set to hold two facilitators and eight slave agents.

The Rules Manager is responsible for maintaining the behavioral rules for each facilitator. Every time a new facilitator is ready for execution it informs the instruction manager and creates a list of behavioral rules that is maintained until the termination of the facilitator agent. The Instruction Manager constantly checks for new arrivals of agents in the network in order to allocate to any new agent entity specific instruction(s) by retrieving the next executable instruction(s). Following the Agillaengine example, the instruction manager is allocated up to 440 bytes, 1 byte per instruction.

The Facilitator Manager is responsible for all the tuple space and reaction operations. The facilitator manager dynamically allocates memory for every tuple insertion. Each tuple can contain up to 4 fields in order to fit the 27 byte payload of a single TinyOS message. It also stores all the reactions registered by a slave agent. During a tuple insertion the facilitator manager checks the registry for a match. When a match is obtained it notifies both the agent manager and the instruction manager. Following the Agilla engine example, the facilitator manager is allocated up to 600 bytes and can recall up to 10 instructions [39].

The In-Motes Engine acts as a virtual machine on top of the TinyOS platform controlling the execution as well as all departures and arrivals of the agent society. Following the Mate and Agilla example the default number of instructions is 4. The engine switches context when an agent in one of the states senses, waits or sleeps. In case of a failure such as package loss it retransmits the packet up to 6 times and then stalls operation. The optimization of the packet retransmission time is the subject of ongoing research.

Each facilitator node currently can support up to four agents and each slave node up to two agents. In-Motes automatically handles the context switching that allows the agents to run in parallel and independently. A facilitator node tuple space is shared by the local slave agents

and it is remotely accessible. However, In-Motes does not address destinations by geographical location rather it uses their node ID. This is reasonable since for a dynamic WSN application in real environments, the spatial location of a node will depend on the initial deployment and can also change during the trial and while the network is active. Also In-Motes, due to the nodes bandwidth and energy limitations, does not support a global tuple space available for every node in the network, rather than a number of local tuple spaces unique for every node. Special instructions are available for the agents of the network to access remote tuple spaces.

Based on the fact that these instructions rely on an id-addressed unicast communication with the specific node, only two messages, a request and reply, are transmitted during a remote tuple space operation. Each facilitator node uses multicast to query the tuple spaces of every slave node [34].

2.13 Chapter Summary

This chapter focused on reviewing existing approaches on using mobile agents for network and spectrum management. However, most of the previous work done on using mobile agents in cognitive radio networks and networks in general suffer from scalability problems because of the use of a single control channel. Control channel-based network management framework suffer from problems such as interoperability, reliability, flexibility and scalability, as the cognitive radio networks become more geographically distributed [19,22]. This research, therefore, proposes an intelligent mobile agent framework for resource distribution and network management in a cognitive radio network environment, using multiple control channels to improve the networks scalability, stability and availability.

Chapter 3

Agent Oriented Spectrum Resource Management

3.1 Overview

This chapter focuses on spectrum resource management in cognitive radio networks using intelligent mobile agents. It starts out with a brief analysis of previous work done on using mobile agents in a cognitive radio network. The system architecture, framework and algorithms proposed by the research are then presented, the aim being to use agents to manage radio resources fairly among multiple users in a cognitive radio network.

3.2 Spectrum Resource Management

The spectrum resource management process in a cognitive radio network consists of four major steps:

a) **Spectrum Detection:** determining the portions of the spectrum currently available, through spectrum sensing. A cognitive radio user can only use an unused or free portion of the spectrum, so they need to intermittently check for the reappearance of the licensed user.

b) **Spectrum Decision:** selecting the best available channel because allocating a free spectrum not only depends on spectrum availability but is also determined based on the characteristics of the channel. In a cognitive radio network, temporarily free spectrum bands will be spread over a wide frequency range and these free bands which are detected through spectrum sensing show different channel characteristics according to the radio environment.

c) **Spectrum Sharing:** coordinating access to this channel with other users since there may be multiple radios trying to make use of the spectrum. Network access should be coordinated in a way that will ensure that radios do not suffer from starvation or collide in the same channel. Since spectrum availability in a cognitive radio network varies over time and space, a dynamic spectrum sharing method needs to be employed by cognitive radios to allow fair resource allocation as well as capacity maximization.

d) **Spectrum Mobility:** effectively vacating the channel when a licensed user is detected or channel condition becomes worse and effectively continuing the communication in another vacant portion of the spectrum. Cognitive radios face several challenges due to the fluctuating nature of the available spectrum as well as the diverse service requirements of various applications. Therefore, it is complicated to maintain a reliable and seamless communication channel. A mobility management framework needs to be devised to support diverse mobility events in cognitive radio networks.

To achieve efficient spectrum utilization, a balanced and integrated communication system is required and the solution proposed in this research is to incorporate spectrum management functionalities with software agents in cognitive radio networks. By using Intelligent Mobile Agents (IMA), the tasks of these four functionalities can be undertaken effectively, and the agents will help in making the most out of the available radio resources, while avoiding interference.

3.3 Analysis of Existing Approaches

Table 2 summarizes few state of the art on using mobile agents in cognitive radio networks and networks in general, with the aim of showing their contributions and their shortcomings.

Table 2: Analysis of Previous Work

Researcher	Contribution	Description
Wu C. et al. 2010 [40]	Spectrum management strategy based on Multi-agent Reinforcement Learning	The approach uses value functions to measure the desirability of choosing different transmission parameters, and enables efficient assignment of spectrum and transmits power by maximizing long term rewards. They employed Kanerva-based function approximation to improve the system's ability to handle large cognitive radio networks and show that the function approximation can effectively reduce the memory used for large networks with little or no loss of performance. Therefore, their proposed reinforcement learning based spectrum management approach can significantly reduce interference to licensed users, while maintaining a high probability of successful transmissions in a cognitive radio ad hoc network.

Ahmed A. et al. [41]	Embedded Agent Modules	Agents are embedded in the radio devices that coordinate their operations to benefit from network and avoid interference to the primary user. Agents carry a set of module to gather information about the terminal status and the radio environment and act accordingly to the constraints of the user application. An intelligent agent is responsible for managing a set of modules on each cognitive radio device. This agent oversees all the operations, including, methods for free band selection, modulation types, management of signal processing algorithms, management of data channels, energy control, security, quality of service etc.
Mark F. et al. 2007 [42]	Game theory-based resource management	Proposes a model in which the communicating nodes share a single collision domain. each node contain a certain number of radio devices, which the proposed model can allocate to the channels. Multiple devices can be assigned to a single channel and the authors claim that the solution congregates to a load-balancing solution despite the non-cooperative nature of users
Kloeck C. et al. 2006 [46]	Bidding strategy	Proposes a bidding strategy whereby radio resources can be allocated to users through prior negotiation. Users and operators are denoted by intelligent mobile agents in the auction. The strategy includes several phases and the agents embedded in the terminals include a learning module that enables to develop a bidding strategy independently.
Xie J. et al. 2007 [19]	Using MAS between multiple WLANs	A technique for radio resource management using Multi-Agent Systems (MAS) between multiple Wireless Local Area Networks (WLANs) that operates in the ISM band. Intelligent agents are placed in the access points, and MAS consists of backbone that connects all the access points in the system. Agents interact with each other through the access point and exchange the state of spectrum availability in their respective coverage area. These interactions allow measuring the availability of spectrum and avoiding interference between different WLANs. This scheme seems more promising as it shows induced management of signaling loads and time delays.
Feng Z. et al. 2006 [21]	Novel distributed intelligent multi-agent system	Presents a self-organized system to manage the coexistence of multi- Radio Access Technologies (RATs) operated by different network providers (NPs). Their multi-agent system is mainly concerned with the coexistence of different RATs and considers not only the satisfaction of local network, but also the satisfaction of local network and neighboring network with overlapping cells.

The reinforcement learning-based approach proposed can significantly reduce interference to licensed users, while maintaining a high probability of successful transmissions in a cognitive radio ad hoc network. The approach on using embedded agent modules can be improved by designing algorithms that will consider the impact of the use of in-band or out-band transmission channels for information exchange between the cognitive radios with the help of agents. In the game theory-based resource management, multiple devices can be assigned to a single channel but this deteriorates the channel's total rate in cases where more devices access it again and again. The bidding strategy approach ignored the quality of service constraints in distributing the spectrum and the multi-agent system performs a macro-control-and-manage function by using control factor and validation mechanism. However, it will be difficult to make the agent generic, open and independent of radio networks.

The need to manage cognitive radio resource and network properly is a compelling factor as proper management will improve the overall performance of the network. In this research, another technique on using intelligent mobile agents for spectrum resource management in cognitive radio networks is therefore proposed. To improve the cognitive radio networks scalability, stability and availability, multiple control channels are used instead of a single control channel. Multiple control channels will increase the throughput of the cognitive radio services and they can work together to provide services for a same part of the cognitive radio network environment. Services for different parts of the cognitive radio network environment can be distributed to multiple control channels, thereby, dynamically balancing the workload on the control channels. The system functions are divided into independent tasks and the tasks are modeled as intelligent mobile agents which are not bound to any fixed node. Therefore, it can autonomously migrate or clone at any suitable cognitive radio node to distribute spectrum more pervasively, to ensure real-time performance and avoid collisions and starvation.

3.4 IMACRN: System Architecture

The developed system, Intelligent Mobile Agent for Cognitive Radio Networks (IMACRN), is an intelligent mobile agent middleware for cognitive radio networks and is based on the injection of mobile agents in the network that can migrate to other nodes or clone to perform

specific tasks. By doing so, each agent is given a certain degree of perception, cognition and control, forming the basis of its intelligence. A Mobile Code Daemon (MCD) runs within a Java Virtual Machine (JVM) on each network component. The MCD receives digitally signed mobile agents and performs authentication checks on them before allowing them to run on the network component. Java is used because it supports code mobility. While resident in the network, mobile agents access managed resources through the Virtual Managed Component (VMC). The VMC provides the get, set, event and notification facilities with an access control list mechanism being used to enforce security. VMCs are designed to contain Managed Information Base (MIB) and vendor-related information.

A Migration Function (MF) provides transport from one Network Component (NC) to another. The Mobile Code Manager (MCM) manages the agent lifecycle while present on the Network Component (NC). The agents solve problems by moving over the nodes and links in the network and interacting with "chemical messages" deposited in that network. These messages are stored within virtual managed components (VMC) and are the principal medium of communication used between groups and individual agents.

Chemical messages are used for communication instead of raw operational measurements from the network in order to separate measurement from reasoning. Also, chemical messages drive the migration patterns of the agents, the messages intended to lead agents to areas of the network which may require attention. These architectural components are given diagrammatically in Fig. 3.1.

3.4.1 IMACRN System Tuple

Agents in IMACRN system can be described as a tuple: $\mathbf{A} = (E, R, C, MDF, m)$

Where:

> ***Emitters (E):*** *are the means by which the local chemical message environment is changed and sensed respectively. Emitters are used to generate chemical information or messages that are deposited where the agent is currently located.*

Receptors (R): *Receptors are used to sense information or chemicals that are present in the agent's local environment and chemical changes that occur in it. Both emitters and receptors have rules associated with them in order that the agent may reason with information sensed from the environment and the local state stored in memory.*

Chemical Messages (C): *defines the set of chemical reactions associated with an agent and these reactions represent the way in which sensed messages can be converted to other messages that can, in turn, be sensed by other agents within the network.*

Migration Decision Function (MDF): *is a function or rule set that is used to determine where an agent should visit next. It is intended to drive mobile agents migration. Its intention is to allow an agent to migrate to different directions and low or high concentrations of the chemicals that an agent can sense, either probabilistically or deterministically.*

Memory (m): *the memory associated with an agent stores the information or chemicals and their concentrations that are held internally to the agent. It is associated with each agent in order that the state can be used in the decision-making processes employed by the agent.*

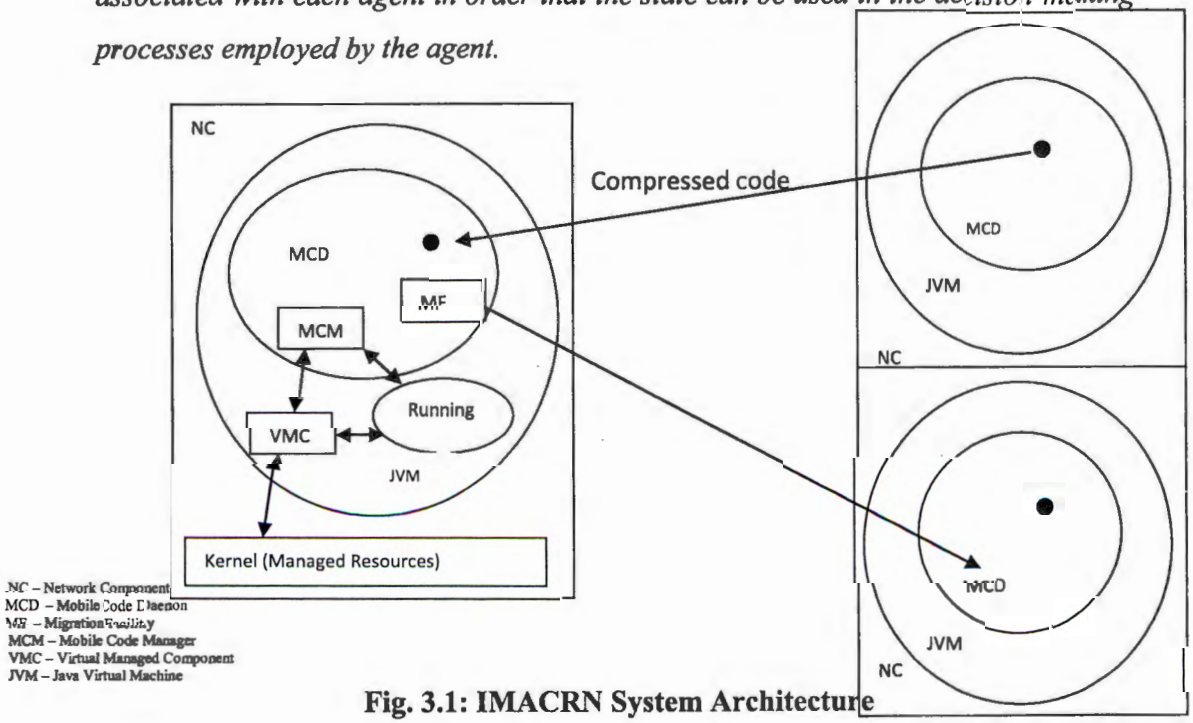


Fig. 3.1: IMACRN System Architecture

3.4.2 Using JAVA

The code in a mobile agent is required to be platform independent so that it can execute at any remote host in the heterogeneous network environment. Some of the scripting languages used to support mobile agents coding are TCL, Java, Perl, C++ and XML (extended markup language). Java is considered to be a more suitable agent development language for the IMACRN system because of its security and portability features in heterogeneous environment. It provides a heterogeneous and distributed architecture which is well suited for the intelligent mobile agent framework. It is interpreted and, therefore, platform independent, so programs can easily migrate from machine to machine. JVM supports built-in security mechanisms for code validations and authentication. Java is object oriented and offers delayed, dynamic binding through the class loader.

JAVA is, therefore, used for IMACRN to ensure interoperability on any network device, regardless of the underlying hardware platform or operating system. IMACRN are JAVA objects capable of migrating between hosts, where they execute as separate threads and perform their specific tasks. They are supplied with an itinerary table, a data and signal folder, where they store collected information. The system architecture consists of distributed mobile agents working cooperatively in supporting and managing the cognitive radio network environment. The services provided by the mobile agents include resource discovery, node monitoring, scheduling of agents and resource sharing. The mobile agents are autonomous and have the ability of migrating among hosts to maximize resource utilization.

3.5 IMACRN Spectrum Resource Management Framework

Basically, IMACRN system design is based on five agent activities to take care of spectrum detection, spectrum decision, spectrum sharing and spectrum mobility. These agent activities are shown in Fig. 3.2. The agent activities are described as follows:

i) Spectrum Sensing Agent (SSA): The function of SSA is to sense the radio spectrum holes and continuously monitor the primary user signals. SSAs cooperatively sense the channel

and measure it against the threshold. The sensing is performed by considering a real-time dynamic environment, because it is not obvious at what time a spectrum band is occupied or when it is free. Factors such as primary user's signal power and associated noise, spectrum traffic, sampling time and intervals are all kept into consideration.

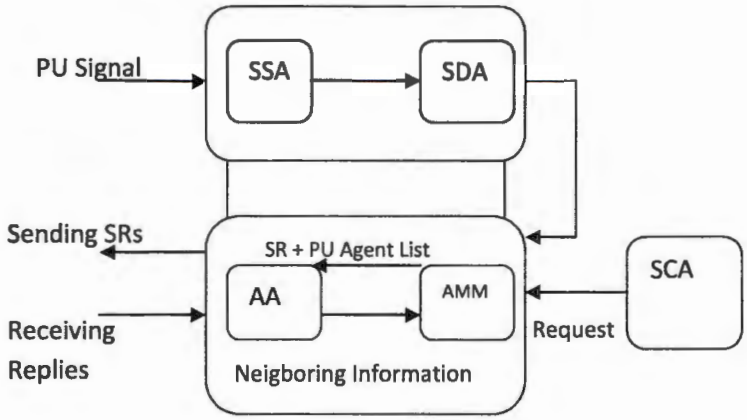


Fig. 3.2: IMACRN System Activities

ii) Spectrum Decision Agent (SDA): The SDA characterizes the spectrum hole and its function is to arrange or divide the spectrum hole information received through the SSAs according to channel information and capacity.

iii) Secondary Consumer Agent (SCA): The SCAs sends spectrum request messages in the form of Call for Proposal (CFP) to the Agreement Agent, whenever a secondary user wants to use a portion of the spectrum. The message is of the form: req(s,t), where *s* is the amount of spectrum needed by the secondary user depending upon its application in use, for a time duration *t*. SCAs function is also to coordinate or share the spectrum amongst secondary users in the network after it has been acquired from the primary user.

iv) Agent Memory Module (AMM): AMMs gets the primary user's signal characterization from SDAs and stores in its database. This list is not permanent, rather it is updated and maintained on regular time intervals. This module also serves as database for available spectrum and their characteristics. Based on inputs from SDA and AMM, SSNAs prepare a Spectrum Request (SR) message

CFP(SUID, s, t, d)

Where SUID = Secondary User Agent Identification

s = Amount of spectrum needed by the secondary user

t = The desired time limit for utilizing the spectrum

d = Deadline to receive the primary user's reply

Having received the SRs, the interested primary user agents send their replies to the corresponding secondary user agents. The reply is of the following form:

REP(PUID, s,t,p)

Where PUID = Primary User Agent Identification

s = Amount of spectrum the PU is willing to give to corresponding secondary user

t = The proposed spectrum holding time

p = The price the primary user is willing to receive

The receiving secondary user locally sorts the replies and sends an 'Accept' message to the primary user with the most suitable reply. This information is also sent to SU-AMM to store for future interactions. Spectrum sharing is then started based on agreed terms from both sides. The primary user can still respond to further SRs if it has other unused spectrum portions and wants to share it.

Besides SR creation, secondary user-AMM (SU-AMM) maintains the neighboring primary user's information received through frequent interaction between the agents, along with a list of previously received requests. This information includes the leaving and joining of neighboring nodes in a network and their current spectrum status and it helps an SSNA to create a more precise Spectrum Request (SR). Uniformly, the primary user-AMM (PU-AMM) functions almost in the same manner by maintaining the neighboring secondary user's arrivals and departures information and a list of their previous spectrum demands.

v) Agreement Agent (AA): AAs manage the cooperation between primary and secondary users for spectrum sharing. After the reception of an SR message from SCA, the Secondary User Agreement Agent (SU-AA) sends the received SR to the currently available primary user agents. The primary users are considered to be available if they still exist in the

corresponding secondary users' neighborhood with their spectrum portions. The PU-AA chooses the most suitable and appropriate SR and sends the reply. Finally, the appropriate agreement for both the primary and the secondary users is the one which is profitable and maximizes their utility values. On average, the utility for a primary user is the price paid by the secondary user agents for their spectrum utilization divided by the amount of spectrum it has shared for the respected time period. A secondary user agent's utility is represented as its spectrum usage for the required time divided by the corresponding price paid to the primary users.

3.5.1 IMACRN-Based Spectrum Detection

Dynamic Spectrum Access (DSA) by cognitive radio-enabled secondary devices is one of the promising approaches to increase utilization of underutilized licensed spectrum bands. However, DSA approach requires that the secondary users should not violate any acceptable interference bounds specified by the primary users. Therefore, the main challenges involved in devising DSA scheme for cognitive radio devices are as follows:

a) The cognitive radio nodes should be able to identify the white spaces in the spectrum and utilize them without interfering with the primary user.

b) The DSA scheme should minimize the channel sensing (and therefore, the energy consumed in the sensing operations) by the secondary node.

c) Cognitive radios cannot transmit while sensing the channel, which undermines the goal of DSA because spectrum is wasting and, therefore, spectrum efficiency is decreased.

In practical multi-user environments, cognitive radio operation is governed by interference tolerance and sensing limits at the primary and secondary users. The interference limits at the primary and secondary users indicate the amount of protection needed at each primary and secondary user from the multi-user interference to maintain a certain rate. On the other hand, the sensing limits (minimum SNR needed for detection) at the secondary users reflect the amount of protection that each secondary user is individually able to provide to the primary users. In these scenarios the key is to strike a balance between the two conflicting goals:

minimizing the interference to the primary users, and maximizing the performance of the entire system.

To overcome these detection challenges, the Intelligent Mobile Agents (IMAs) strategy is devised to help in reliably detecting idle channels, thus limiting the number of secondary users sensing the channel. The scheme ensures non interference to the primary user and radios can concentrate on transmission while the agents sense the channel for the reappearance of the primary user.

Spectrum Sensing Agents (SSA) in IMACRN are encoded to sense and detect primary user signals and send the information to the cognitive radio network. The function of SSAs is to sense the radio spectrum holes and continuously monitor the primary user signals. A predefined set threshold *Y*, is also input in the code for use in measuring the observed signal. This will help the radios to concentrate more on using the available spectrum without having to intermittently check for the presence of the primary user. The agents will individually or collaboratively detect active primary user transmissions over the band, and decide if the sensing results indicate that all the primary user transmitters are inactive at that band.

To buttress the points made, let us represent spectrum sensing by IMACRN agents using the algorithm below:

```
  let;   Y = a set threshold
Where    Y = Energy observed on the primary user signal
  let;   s = result obtained from sensing the spectrum
   if    s > Y
 then    H1 = Primary User is Present
  else
   if    s < Y
 then     H0 = Primary User is Absent.
// The set threshold Y will be used to determine and measure the reliability of the collected results. When the collected signal Si exceeds the threshold Y, decision 1 will be made which assumes that the primary user is present; otherwise, decision 0 will be made. The decision Di of the agents is then given by:
Di = 0; where 0 < Si < Y
Di = 1; where Si > Y
```

The flowchart depicting the spectrum detection algorithm is given below:

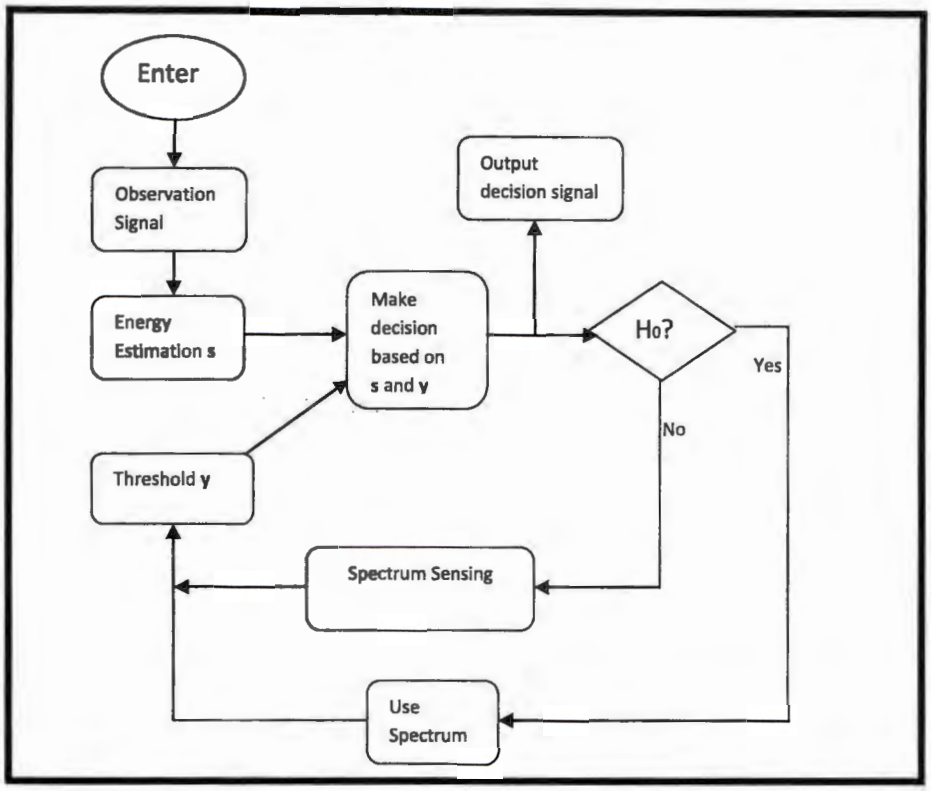


Fig. 3.3: Agent Spectrum Detection

3.5.2 Agent Spectrum Decision

The free spectrum bands detected through spectrum detection show different characteristics according to radio environment. Since cognitive radio networks can have multiple available spectrum bands having different channel characteristics, they should be capable of selecting the proper spectrum bands according to the application requirements; this is called Spectrum Decision. IMACRN Spectrum Decision Agents (SDA) characterize the available spectrum hole and each spectrum band is characterized based on not only local observations of cognitive radio users but also statistical information of primary networks. Through the local measurement, SDAs can estimate the channel conditions such as capacity, bit error rate (BER), delay and jitter. After the spectrum characterization, the best and appropriate spectrum band is chosen.

The algorithm for IMACRN spectrum decision is given as follows:

// Option 1: In IMACRN, channel can be characterized based on capacity C. this is calculated using Shannon's theorem:

$$C = B\log_2(1 + \frac{S}{N})$$

Where C = channel capacity

B = bandwidth of the channel

S = average signal power over the bandwidth

S/N = Signal-to-Noise Ratio (SNR)

[Secondary user agents characterizes each primary user on the basis of capacity]

For each i{ i ∈ PU} do

Evaluate (SNR(i))

[SNR: is the primary user's signal to noise ratio obtained through SSA]

Evaluate (B(i))

[B: is the bandwidth of the primary user given by SSA]

$C(i) = B(i) \log_2 [1 + SNR(i)]$

[C: is the capacity calculated using Shannon theorem]

If acceptable

then D_1

else D_0

End For

// Option 2: IMACRN characterizes spectrum based on the whole spectrum band information and channel capacity. Here the agent needs to check each parameter separately and then go to the next one. When all parameters have been evaluated and the result is satisfactory depending on the application that needs to use it, decision D_1 will be made, otherwise, decision D_0 will be made.

if s < Y

then H_0

Check spectrum band information (b)

where b = 1 to 11

1) operating frequency; 2) bandwidth; 3) interference level; 4) channel error rate; 5) path loss; 6) link layer delay; 7) wireless link errors; 8) holding time; 9) bit error rate; 10)delay; 11) jitter

if b = Acceptable

then D_1

else D_0

end if

The flowchart for spectrum decision is as shown in Fig. 3.4:

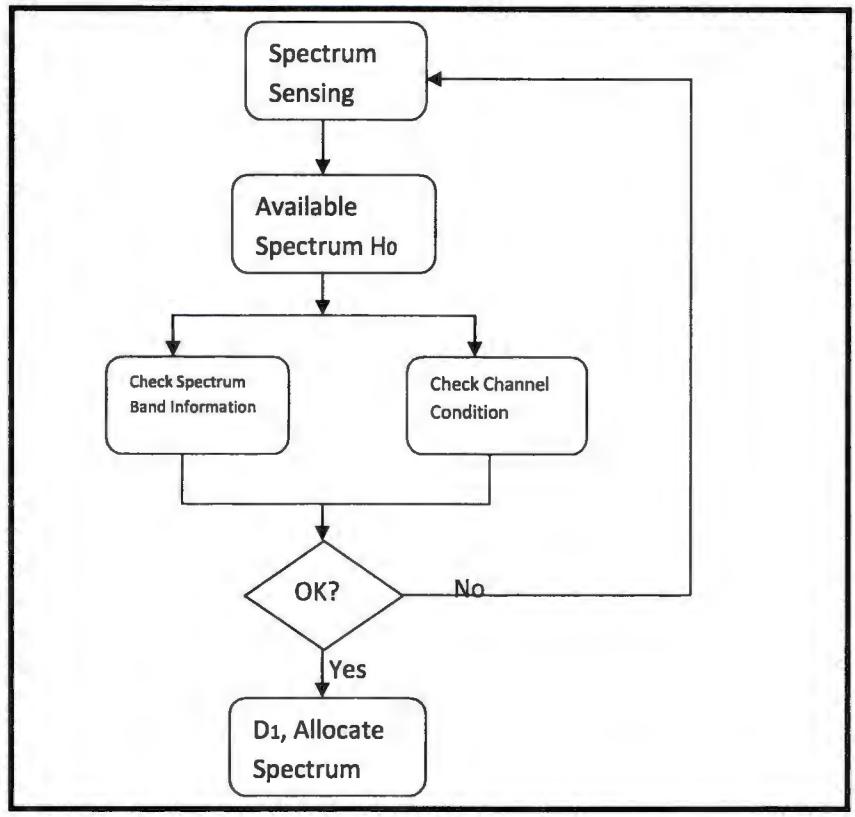


Fig. 3.4: Agent Spectrum Decision

3.5.3 Agent Spectrum Sharing

One of the key issues in cognitive radio networks is to avoid device level collisions and interferences while maintaining efficient spectrum usage. Spectrum holes are not utilized by the secondary users for free. They come at a price, and therefore, they have to first negotiate with the primary user or primary user agents before using the spectrum. Also there are multiple agents, primary users and secondary users in the network. Therefore, at any time, multiple primary users can have multiple free spectrums available to secondary users and multiple secondary users may also need to use spectrum at the same time.

Spectrum sharing and access are important issues facing opportunistic communication in multi-user cognitive radio systems. Because of the presence of user priority (primary and secondary), they pose unique challenges that are not faced in conventional wireless systems.

Spectrum availability varies over time and space in a cognitive radio network. Therefore, a dynamic spectrum sharing capability is required to allow for fair resource allocation as well as capacity maximization and also to avoid starvation problems. Mobile agents should coordinate access to the free channel in a cognitive radio network in a dynamic, fair and organized manner. A cognitive radio network is made up of primary users and secondary users coexisting to utilize the available spectrum. Therefore, IMACRN spectrum sharing strategies are divided into two schemes: a) primary user agent and secondary user agent spectrum sharing agreement scheme, and b) secondary users local spectrum sharing scheme.

a) PU Agents and SU Agents Spectrum Sharing Agreement Scheme

Improvement in spectral efficiency can be achieved through efficient primary-secondary user sharing, where the primary user has usage rights that make quality of service guarantees possible, and the secondary systems operate without causing harmful interference to the primary user. In cooperative sharing, the primary and secondary users interact, whereby the secondary device may ask the primary device for permission to use the spectrum before transmitting. This is advantageous to both the primary and secondary users because it creates an opportunity for the primary user to demand payment and create an opportunity for the secondary user to be guaranteed good quality of service.

IMACRN is a multi-agent system, in which agents are deployed over primary and secondary devices. Individual secondary user agents should send messages to the appropriate neighbouring primary user agents whenever needed and subsequently, the related primary user agents should reply to these agents in order to make a spectrum sharing agreement.

The spectrum sharing process for a secondary user starts by getting the characterization results and the user requirements and continues until the sending of SRs and receiving the replies. The process ends either by having an agreement or a disagreement. For a primary user, the process is the same, by first analyzing the received SRs, sending the replies and finally ending the process either by receiving an 'accept' or a 'reject' message from the

conforming or requesting secondary user. After the reception of an 'accept' message, the spectrum sharing is started between secondary users and primary users, and it continues until the free spectrum is completely utilised and the respected price is paid. If a secondary user receives more than one proposal which are equally satisfactory, then the decision will be made on a first in first out (FIFO) basis.

The Algorithm for secondary user behaviour during spectrum sharing is as follows and its flowchart shown in Fig. 3.5.

// Secondary user agents characterizes each primary user on the basis of capacity
For each i{ i ∈ PU} do
Evaluate (SNR(i))
// SNR: is the primary user's signal to noise ratio obtained through SSA
Evaluate (B(i))
// B: is the bandwidth of the primary user given by SSA
C(i) = B(i) log2 [1 + SNR(i)]
// C: is the capacity calculated using Shannon theorem
End For
// Sending of SR messages
If PU ! = { }
For each i ∈ PU
Send SR(SUID, s, t, d) to PU(i)
End For
End If
// L is a list for saving received SR replies
For each received reply 'r' do
Characterize r using $\frac{s(r)\times t(r)}{p(r)}$ and add to list L
Where p(r) is the price
End For
If L = { } and the deadline to receive replies has expired
Recreate SR
Else if L = {i} where i is the only element in L and the deadline for reply reception has expired
Send an accept message to i
Else
Send 'Accept' to primary user corresponding to best and favourable reply
Send 'Reject' to all other primary users
End If

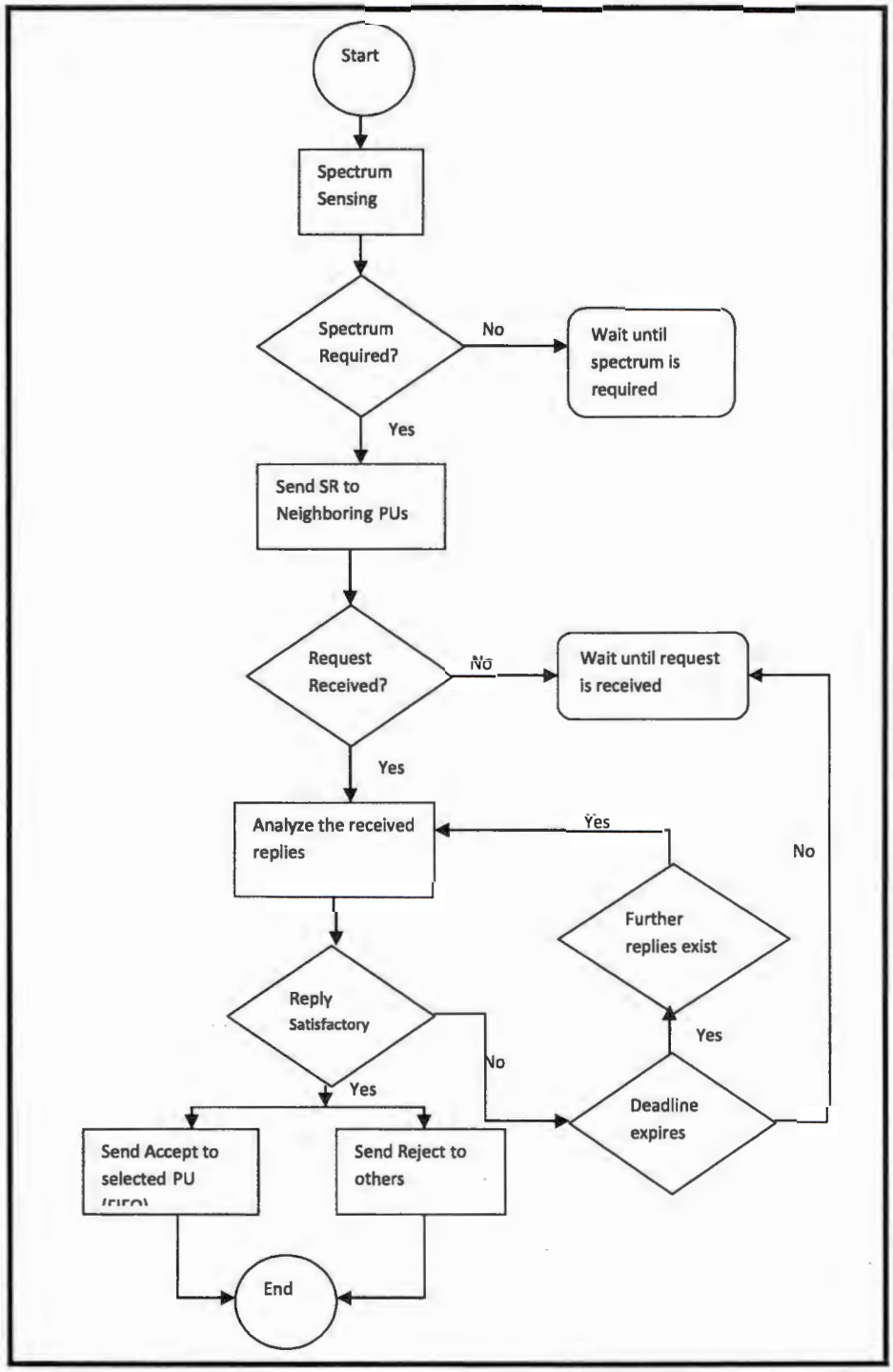


Fig. 3.5: Flowchart for Secondary User Spectrum Sharing Behaviour

The Algorithm for primary user behaviour during spectrum sharing is shown on the next page and the flowchart is shown in Fig. 3.6.

While Busy signal = false do

If received message = SR

For each received SR 'q' do

Characterize q using $\frac{p(q)}{s(q)\times t(q)}$ and update K

[K is a list for saving received SRs]

Where p(q) is related price according to required spectrum

End For

For best SR in K do

Construct a reply (PUID, s, t, p) and send it to corresponding secondary user

End For

End If

If received message = accept

Start spectrum sharing with selected user

End If

If received message = reject or some unused spectrum portions are still available

Continue analyzing further SRs for spectrum sharing

Else

Set Busy signal = true

End If

End While

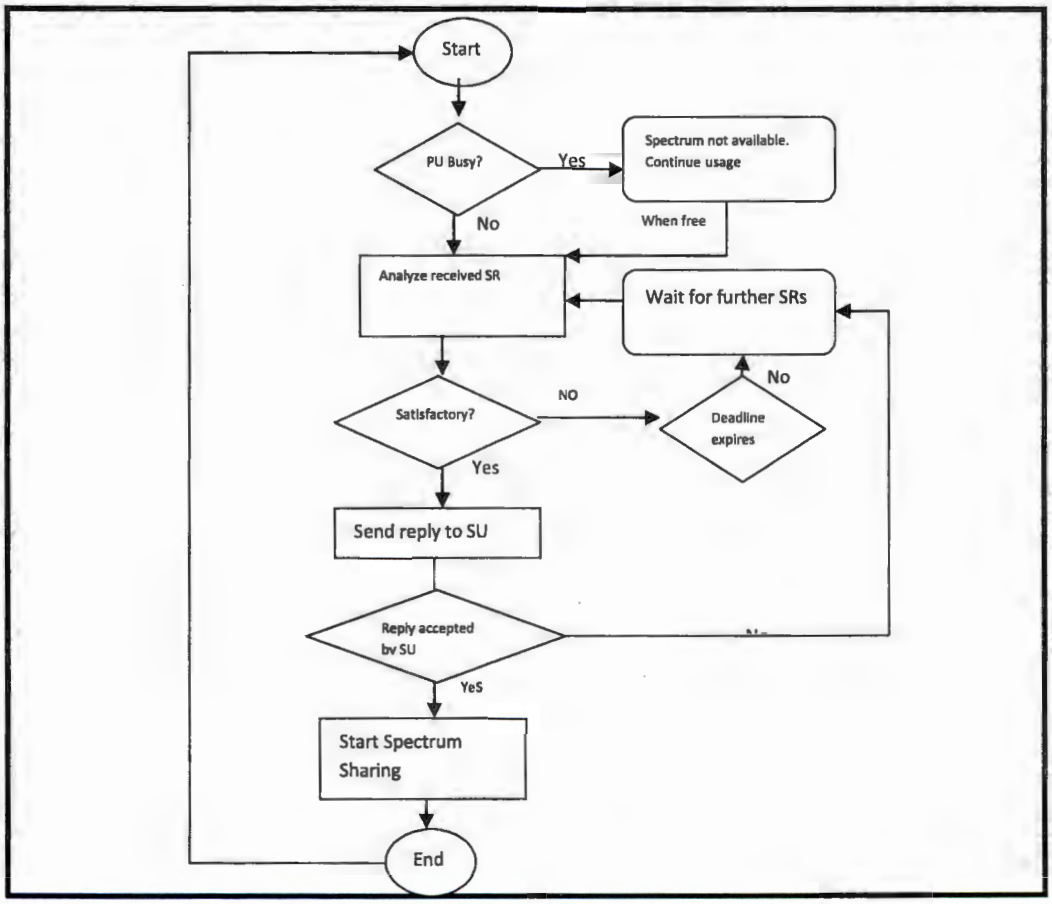


Fig. 3.6: Primary User Spectrum Sharing Behaviour

b) Secondary Users Local Spectrum Sharing Scheme

On acquiring the spectrum from the primary user agents, secondary user agents also need to coordinate the allocation and distribution of spectrum amongst the competing secondary users. A priority-based spectrum strategy is used in IMACRN for spectrum sharing amongst cognitive radios, whereby radios with high priority are considered first and those with low priority are considered last. Spectrum is shared through the use of a Prioritized Queuing Model:

- Class 1 requests: have the highest priority to receive a channel. These are handoff requests. This strategy tries to minimize the probability of disrupting ongoing communication.
- Class 2 requests: are the channel requests by originating or starting communications.
- Class 3 requests: are requests for channel reassignment that are not urgent.

The algorithm for secondary users local spectrum sharing is as follows:

```
if
  D1
then
  allocate spectrum to
   class 1;
   class 2;
   class 3;
 allocate spectrum
```

This prioritized queuing scheme ensures that radios on handoff, which are radios that needs to vacate a channel they are using already due to the reappearance of the primary user or channel degradation, can continue communication seamlessly on the next available channel. This way, good quality of service for the radios on handoff is ensured.

Channel requests by radios that want to start communication are considered next, and the channel can be allocated to them immediately if no class 1 requests are made. Channel can also be allocated to them immediately if there are other idle channels in the spectrum

database. Finally requests from radios that just want to switch channels for no urgent reasons are considered last. They can also get a channel immediately if there are no class 1 or 2 requests or if there are idle channels in the spectrum database

The flowchart for the prioritized sharing scheme is given in Fig. 3.7:

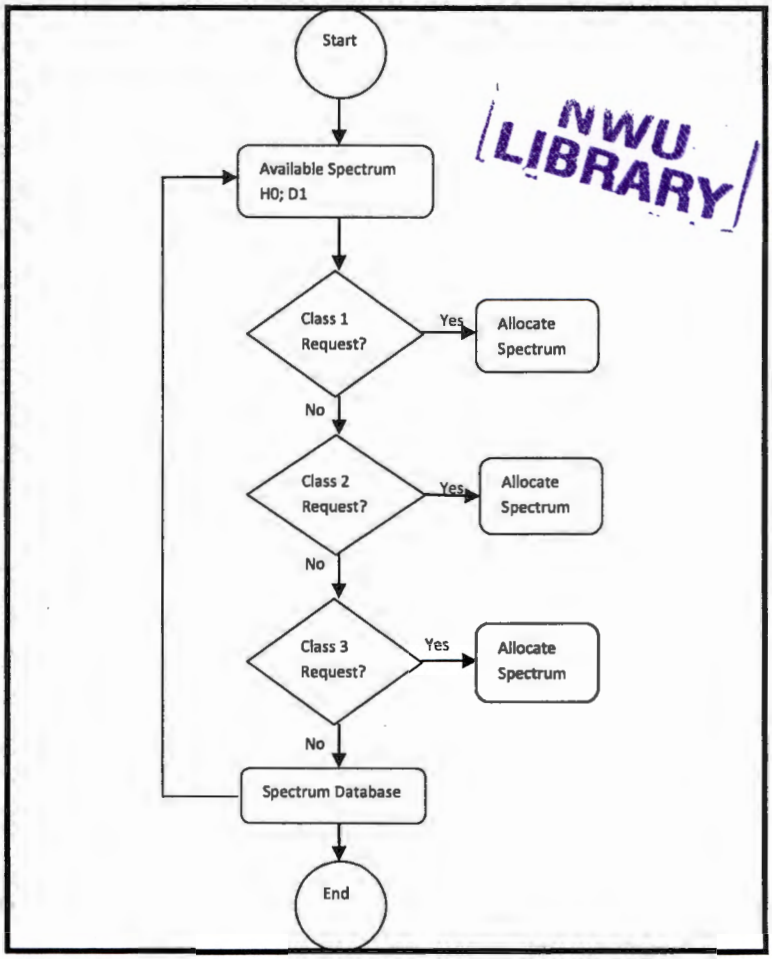


Fig. 3.7: Prioritized Queuing for Spectrum Sharing Among Secondary Users

3.5.4 Agent Spectrum Mobility

Cognitive radios give users the ability to switch channels and make use of dynamic spectral opportunity. However, switching channels takes time and may disrupt the quality of a user's transmission. When a cognitive radio user's channel becomes unavailable, they may switch

channels or they may tune down their transmitter power. Channel opportunities of cognitive radios depend upon the activities of the license holders. When a license holder is inactive, the channel is available to users. When the license holder is active, cognitive radio users must cease transmission in the channel or ensure the amount of interference they cause is below a certain threshold in spectrum overlay networks.

To utilize channel opportunities, cognitive radio users need spectrum agility, which means they need the ability to switch channels quickly to avoid significant interference with the license holder, to make use of spectrum opportunities and also reduce the amount of interference they cause for each other. In practice however, switching channels will inevitably cause disruption as it leads to delay, potential packet loss and the possibility of a broken transmission.

Channel switching has been extensively studied in cognitive radio networks, particularly in cases where users have stochastic channel information. In the IMACRN system, agents maintain a database of information on the Agent Memory Module (AMM), about channel availabilities over the coming time slots. Therefore, users have a foreknowledge about channel availabilities over the coming T time slots. This enables them to formulate a spectrum mobility plan detailing when they should switch channels, to maximize their quality of service.

Spectrum mobility occurs when the primary user reappears or the current channel condition in use becomes worse. Cognitive radio networks face several challenges due to the fluctuating nature of the available spectrum, as well as the diverse service requirements of various applications. Furthermore, the primary user may appear while the channel is still being used by a cognitive radio, which will require the cognitive radio to tune down its transmitter power value or vacate the spectrum and continue its communication in another available channel.

The algorithm for spectrum mobility is given as:

```
// Mobile agents should ensure that cognitive radios effectively vacate the channel when a licensed user is
detected or when the channel condition becomes worse.
  if
    s > Y
  then
    H1
  action
     reduce power value;
     or vacate spectrum
  if
     channel condition degraded
  then
    H1
   vacate spectrum
check spectrum database
switch to the next available spectrum on the database
continue transmission
```

3.6 Agent Migration and Cloning

For the system workload to be effectively decentralized, the IMACRN software system is modeled as a group of collaborative intelligent agents. Each agent is a lightweight software component which provides a certain service for controlling the system signal and data flows. Intelligent mobile agents collaborate autonomously with each other to maintain the cognitive radio network. They can automatically migrate or clone to any qualified participating node to dynamically distribute the workload and ensure high quality cognitive radio network performance. The mobility of agents in IMACRN has two forms: agent migration and agent remote cloning Fig. 3.8.

Agent Migration: is a mobile action to transfer an agent with its run-time state from one host to another, without degrading the real-time performance of the cognitive radio network application. Agents migrate from overloaded nodes to less loaded ones so that they will have sufficient computing and network bandwidth resources to provide their services. Agents can

also migrate to nodes close to user nodes to enhance their service quality. Agent migration provides a method to distribute the system services to multiple hosts. In IMACRN, the state and the code are transferred together only when an agent is migrating for the first time because the JAVA class loader stores every loaded class on a local code cache.

Agent Remote Cloning: agent remote cloning is a mobile action to create a new instance of an agent with its run-time state at another host. After the remote cloning, the cloned agent provides the same service corporately with the original one. A cloned agent also has the same characteristics as the original agent and can perform tasks which were delegated to the original agent. An original agent can clone multiple agents to migrate to other nodes in the network and execute tasks on its behalf. Agent remote cloning provides the mirroring service at a remote host, which can enhance the throughput of the service and also provides a method to distribute the workload of a same service to multiple hosts.

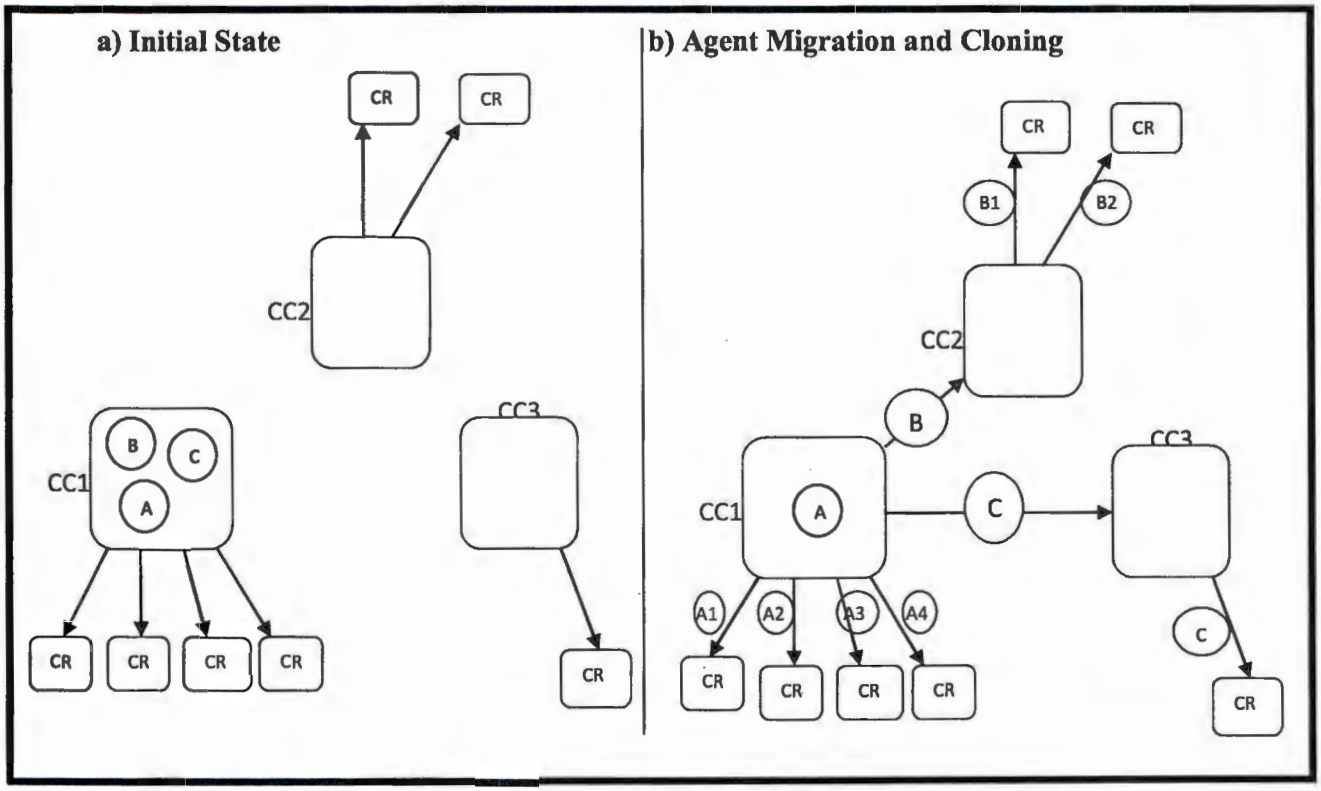


Fig. 3.8: IMACRN Migration and Cloning

Agent A, B and C ran at a busy control channel CC1 (Fig8a), to avoid potential bottleneck emerging at the busy CC1, B and C automatically migrate to other less loaded control channels CC2 and CC3. A automatically clones new instances A1, A2, A3 and A4 respectively at four trusted cognitive radio nodes CR and B also clones new instances B1 and B2 (Fig 8b).

3.7 Agent Deployment

In IMACRN, tasks are performed by the agents. Because of the dynamic user interaction, the fluctuation of available resources and the ever changing nature of network traffic, the workload of each agent or node vary over time in a nondeterministic way. On the other hand, each agent is an autonomous entity which is self-aware, with their own performance and resource requirements. Three sub-processes are used to deploy mobile agents to nodes in a cognitive radio network. These sub-processes are: mobile action initiation, mobile action reasoning and node discovery.

(i) Mobile action initiation: this determines when it should propose an agent mobile action to avoid the degradation of the cognitive radio network performance.

(ii) Mobile action reasoning: here, when an action is initiated, the mobile action reasoning process begins and proposes which agent should perform a mobile action and what mobile action the agent should perform.

(iii) Node discovery: after the above steps, an agent is selected and it will migrate to the ideal node found. If the node still has agents running on it, the agent will return back to mobile action reasoning. The flowchart for these actions is given in Fig. 3.9.

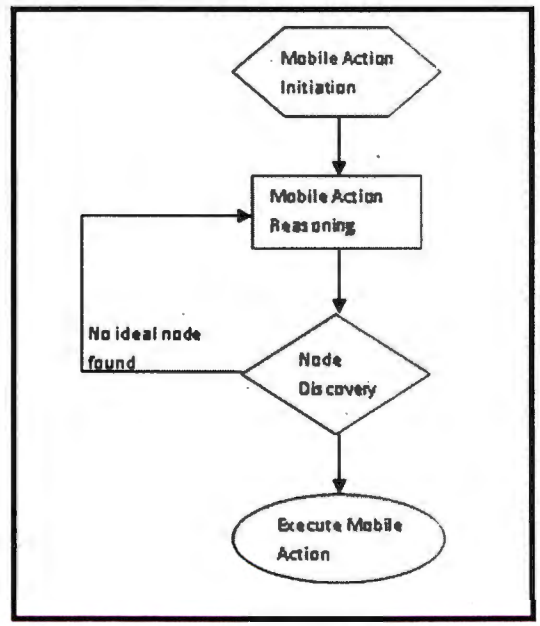


Fig. 3.9: Agent Deployment Process

3.8 Migration Control

Migration of agents to and from the network and the radio nodes need to be controlled to avoid incomplete tasks and also to ensure that a deployed agent's task is performed successfully, even if the agent breaks down. In IMACRN, when starting an agent with a predefined itinerary, a copy is stored at the originating location. A failure at the station or node currently hosting the mobile agent does not necessarily lead to a breakdown of the agent's task, since an up-to-date copy of the agent's itinerary is kept at the originating location and a new agent can be created in order to visit the remaining stations or nodes of the itinerary. This ensures successful and complete execution of tasks.

3.9 Chapter Summary

Spectrum resource management functionalities are important features to realize Dynamic Spectrum Access (DSA) principles. This chapter detailed Java-based Intelligent Mobile Agent-oriented approaches for:

(i) Spectrum detection and decision, whereby a set threshold is input into the mobile agent codes for use in detecting an idle or free spectrum and mobile agents evaluate the channel based on spectrum band information and channel capacity.

(ii) Spectrum sharing, where primary and secondary user agents send messages to each other and negotiate spectrum access. Agents also use a priority-based scheme to share spectrum amongst secondary users.

(iii) Spectrum mobility, in which agents notify the secondary user on the reappearance of a primary user, and indicate need for a transmitter power level change or vacating of the spectrum.

This agent-oriented approach can significantly reduce interference to the primary users and ensures efficient spectrum usage as radios can use the spectrum without having to stop to check for the primary user's reappearance. Fair spectrum sharing amongst primary users and secondary users can be achieved which ultimately leads to good quality of service for both the primary and the secondary users.

Chapter 4

Implementation and Results

4.1 Overview

This chapter reports on the developed intelligent mobile agents in Java Agent Development Environment (JADE) and the results obtained from implementing them. The developed agents were used for spectrum management functionalities in a cognitive radio network and the results were documented.

4.2 IMACRN Agent Creation

IMACRN agent in JADE was created by defining a class extending the jade.core.agent class and executing the setup() method, which includes agent initializations. The actual job an agent has to do is carried out within "behaviours". Fig 4.1 shows the created agent using the above mentioned method.

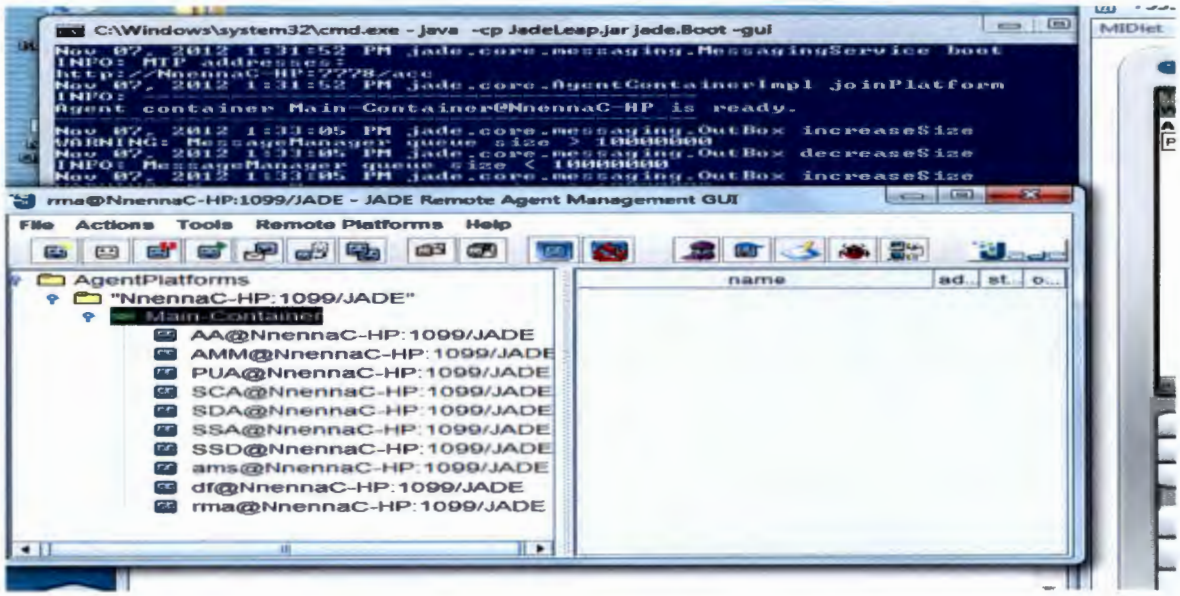


Fig. 4.1: IMACRN Agents in JADE

4.2.1 IMACRN Agent Identifiers

Each agent was identified by an agent identifier (AID), represented as an instance of the jade.core.AIDclass. The getAID() method of the agent class allows retrieving the agent identifier. An AID object includes a globally unique name plus a number of addresses. The name in JADE has the form <nickname>@<platform name>, for instance, in Fig. 4.2, our spectrum sensing agent (SSA) living in a platform called NnennaC-HP had SSA@NnennaC-HP as its globally unique name.

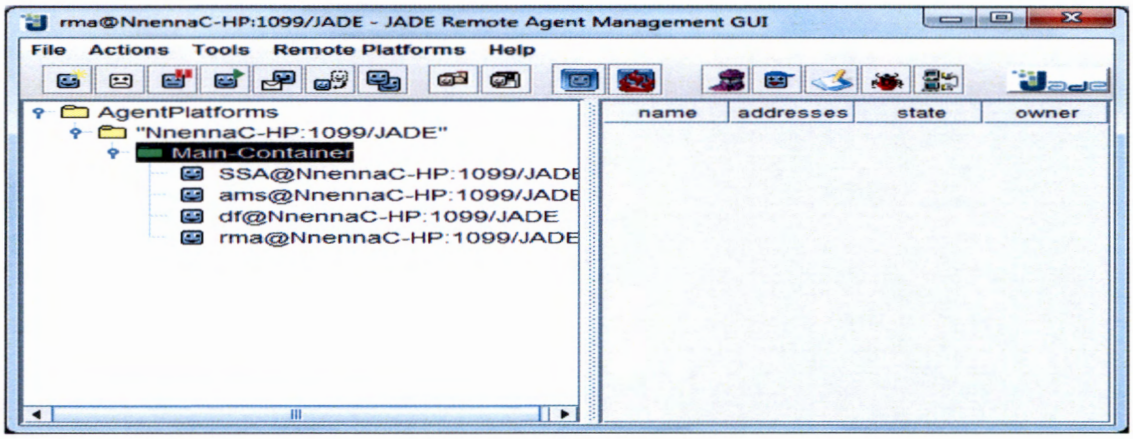


Fig. 4.2: Agent Identifiers (AID)

4.2.2 Agent Communication

The communication paradigm adopted by IMACRN agents was the asynchronous message passing. Each agent has a mail box (the agent message queue) where the messages from other agents are stored. The messages exchanged by the IMACRN agents have a format specified by ACL defined by the FIPA international standard for agent interoperability. This format comprises of a number of fields and in particular:

i) the sender

ii) the receiver(s)

iii) the communicative intention (also called performative), indicating what the sender intends to achieve by sending the message. The performative can be

REQUEST – if the sender wants the receiver to perform an action

INFORM – if the sender wants the receiver to be aware of a fact

QUERY-IF – if the sender wants to know whether a given condition holds

CFP (Call for Proposal), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL – if the sender and receiver are engaged in a negotiation.

4.2.3 Using "Yellow Pages" Services

In the case that SUAs don't know the PUAs and the services they offer, they can use the yellow pages services as shown in Fig. 4.3, provided by the JADE platform to dynamically discover PUAs with the desired services, available at a given point in time.

A yellow pages service allows agents to publish one or more services they provide so that other agents can find and successfully exploit them. The yellow pages services in JADE are provided by an agent called Directory Facilitator (DF) and each platform hosts a default DF agent (whose local name is "df"). Other DF agents can be activated and several DF agents (including the default one) can be federated to provide a single distributed yellow pages catalogue.

4.2.4 Publishing Services

PUAs wishing to publish their services provide the DF with a description including its AID. For each published service, a description is provided including the service type, the service name and a number of service specific-properties. For instance, the PUA in Figure 4.4, will provide information about the capacity and bandwidth of the available spectrum of which it wants to publish. This will make it convenient and easy for any SUA wishing to acquire a particular spectrum of specified values.

4.3 Experimental Results

In this section, various numerical results to evaluate the multi-agent approach are presented. The simulation time was set at 120 minutes. All the simulations were conducted in Java Application Development Environment (JADE), over two PCs with 3.40GHz and 2.30GHz processor and 4GB memory. The parameters used are as shown in Table 4.1.

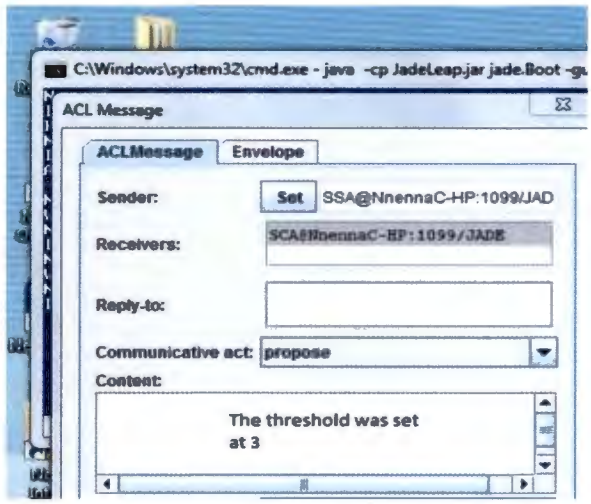
Table 4.1: Parameters for Experiment

Parameters	Value
Size of spectrum portion	4MHz
Simulation time	120 minutes
Max. number of PUs	30
Max. number of SUs	30
Max. number of each type of agent	5

4.3.1 Spectrum Detection

For spectrum detection to take place, two agents must be communicating through messages. The two agents in the experiment are Spectrum Sensing Agent (SSA) in Fig. 4.3(a) and Secondary Consumer Agent (SCA) in Fig. 4.3(b). The agents use a set threshold in their estimation for the presence or absence of the primary user. As can be seen in Fig. 4.3, the spectrum sensing agent and the secondary consumer agent communicated using messages.

For the purpose of this experiment and for communications to take place between agents, the threshold was set randomly at 3, and on sensing a signal at 1, the SSA sent the information to the SCA indicating that the spectrum is available for use. And on receiving the message, the SCA acknowledged receipt by replying as shown in Fig. 4.4. The experiment was performed with a single and later with a multiple number of agents and the results obtained are reported in Table 2.



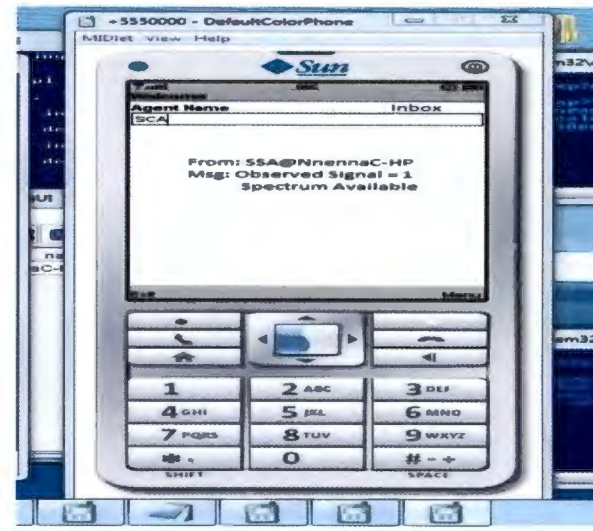


Fig. 4.3(a): Spectrum Detection Message from SSA

Fig. 4.3(b): Message Received by SCA

Fig. 4.4: Acknowledgement Message from SCA

From the results in Table 4.2, it can be shown that agents were able to correctly detect the occupancy state of the channel, when there are multiple numbers of agents.

Table 4.2: Spectrum Detection Data

Number of Channels Sensed	Correct Detection (1 SSA)	Correct Detection (5 SSA)
5	3	5
10	6	8
15	10	13
20	13	17
25	20	23

With results shown in Table 4.2, the correlation between the number of agents and their corresponding number of channels correctly detected is graphically represented in Figure 4.5.

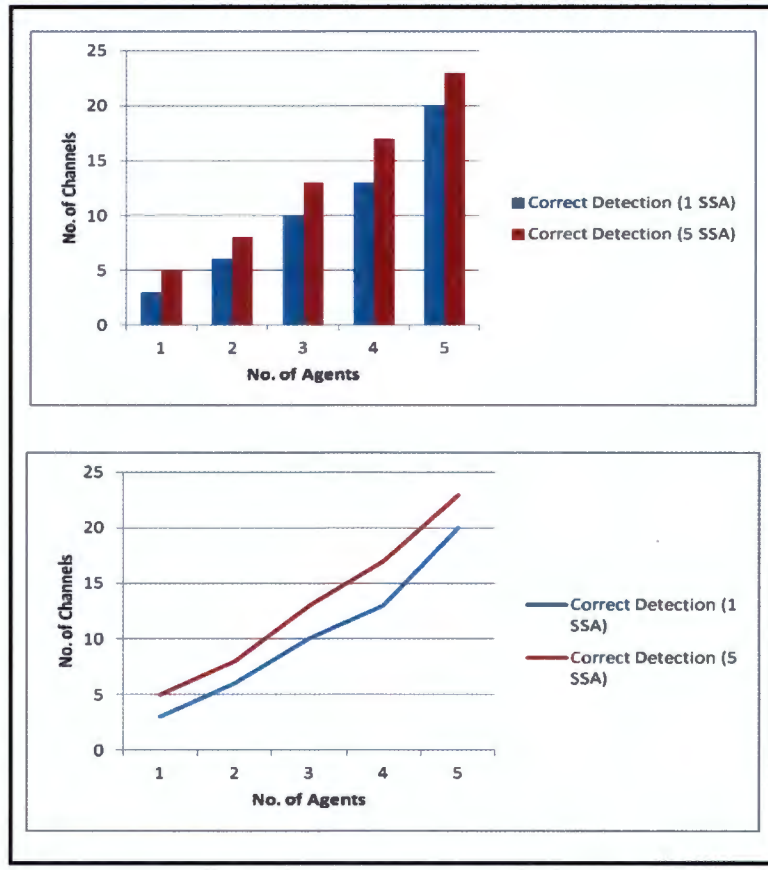


Fig. 4.5: Spectrum Detection Data

4.3.2 Spectrum Decision

The detected spectrum needs to be characterized to ensure that it is in good condition and appropriate for use. The agents were able to correctly decide on the appropriate spectrum band based on the characteristics of the available channels. Here, agents also use messages to pass information to each other. The results obtained based on a single agent decision and a multiple number of agents are as shown in Table 4.3.

Table 4.3: Spectrum Decision Data

Number of Available Channel	Correct Decision (1 SDA)	Correct Decision (5SDA)
5	3	5
10	6	8
15	10	14
20	13	20
25	20	24

The SDA, after characterizing the channel, sent a message to the SCA indicating that the channel is usable, as shown in Fig. 4.6.

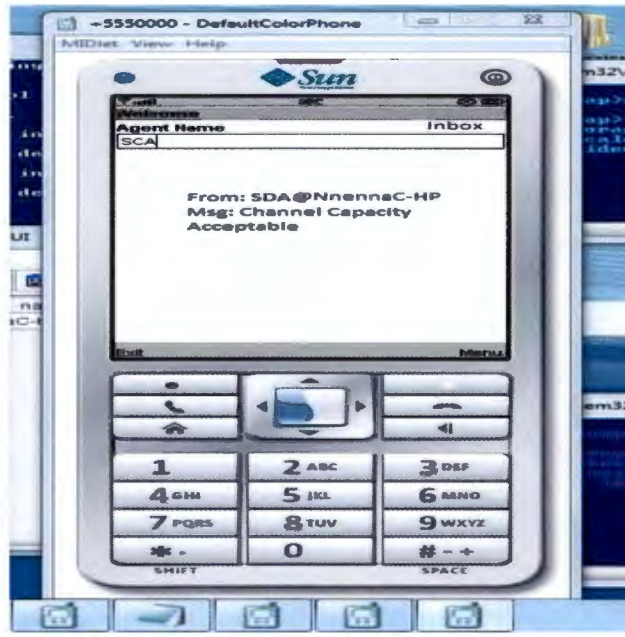


Fig. 4.6: Message from Spectrum Decision Agent (SDA)

Graphical representations of the correlation between the number of agents and decisions made correctly are shown in Fig. 4.7. From the results, it can be seen that multi-agents characterized the channels more correctly than a single agent. This is due to the cooperative nature of IMACRN agents in spectrum detection and decision. Each agent shares its detection and decision observation with other agents through messages, these observations are combined and decision made based on collective agreement.

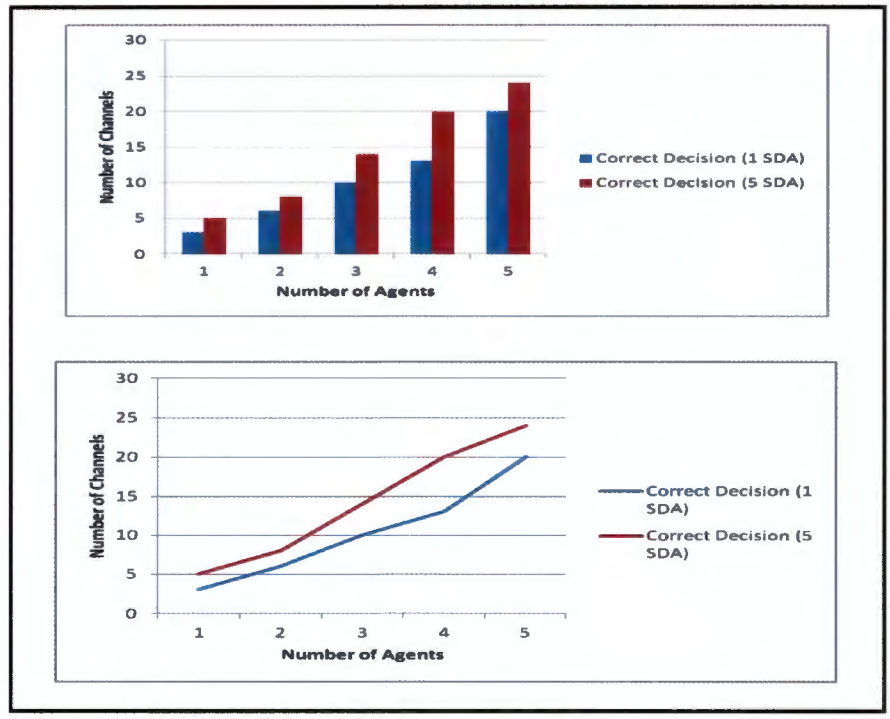


Fig. 4.7: Spectrum Decision Data

4.3.3 Spectrum Sharing

For spectrum sharing, the two agents involved in negotiations are the Secondary Consumer Agent (SCA) and the Primary User Agent (PUA). In Fig. 4.8, the SCA sent a Call for Proposal (CFP) message to the PUA for spectrum sharing. The message indicates the agent identification, the size of spectrum needed, the duration the spectrum will be used and the time it expects reply from the PUA. The PUA then sent a reply indicating its identity, the size of spectrum it is willing to share, the holding time of the spectrum and the price in Rand (R). The results obtained for various experiments are as shown in Table 4.4.

Table 4.4: Spectrum Sharing Data

Number of Primary Users	Number of Secondary Users in need of Spectrum	Number of Served Secondary Users
10	20	17
20	25	22
30	30	28

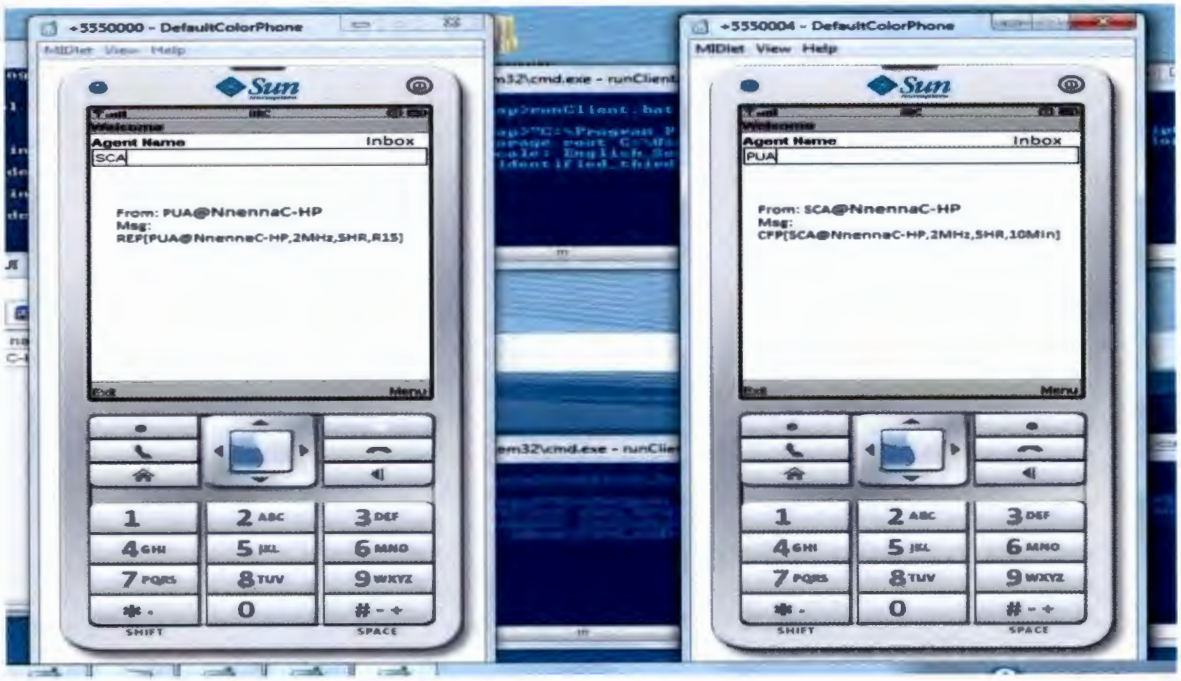


Fig. 4.8: Message Exchange for Spectrum Sharing

On receiving the PUAs proposal, the SCA sends a reply back to the PUA, either accepting or rejecting the proposal as shown in Fig. 4.9.

Fig. 4.9: Accept Proposal Message

The correlation between the number of secondary users in need of spectrum and the number of secondary users served in the end is represented graphically in Fig. 4.10.

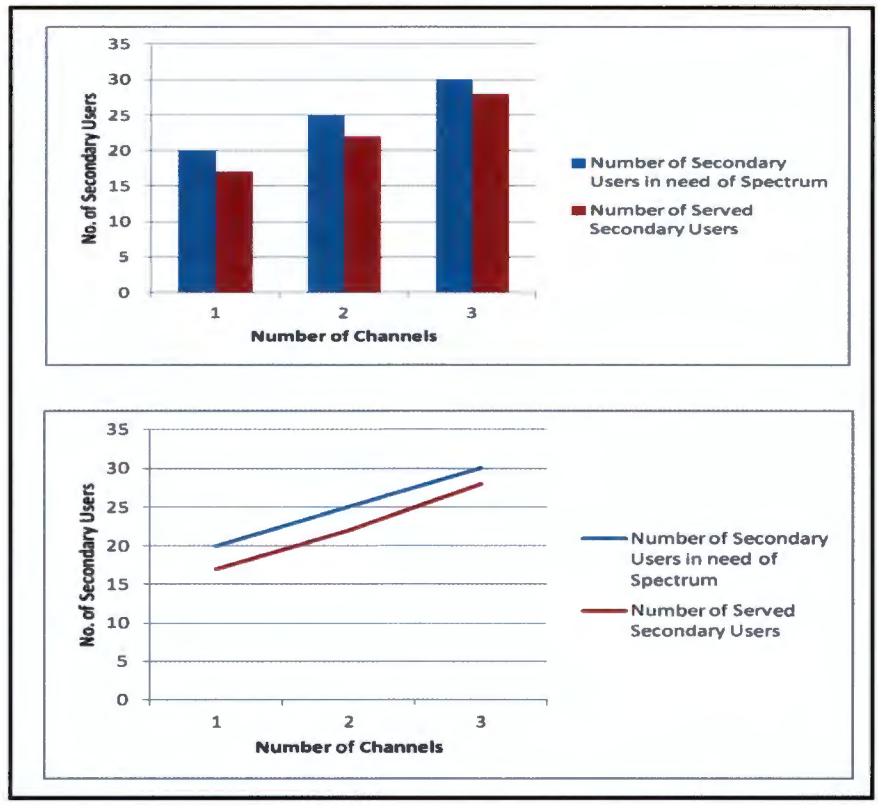


Fig. 4.10: Spectrum Sharing Data

From the results, a large percentage of secondary users wishing to acquire spectrum were served, which indicates that the secondary user agents (SUAs) were able to interact and negotiate successfully with the primary user agents (PUAs) for spectrum sharing.

4.3.4 Spectrum Mobility

For spectrum mobility, once an agent detects the reappearance of primary user, it sends out a message into the network and to the SCA using the channel to vacate as can be seen in Fig. 4.11.

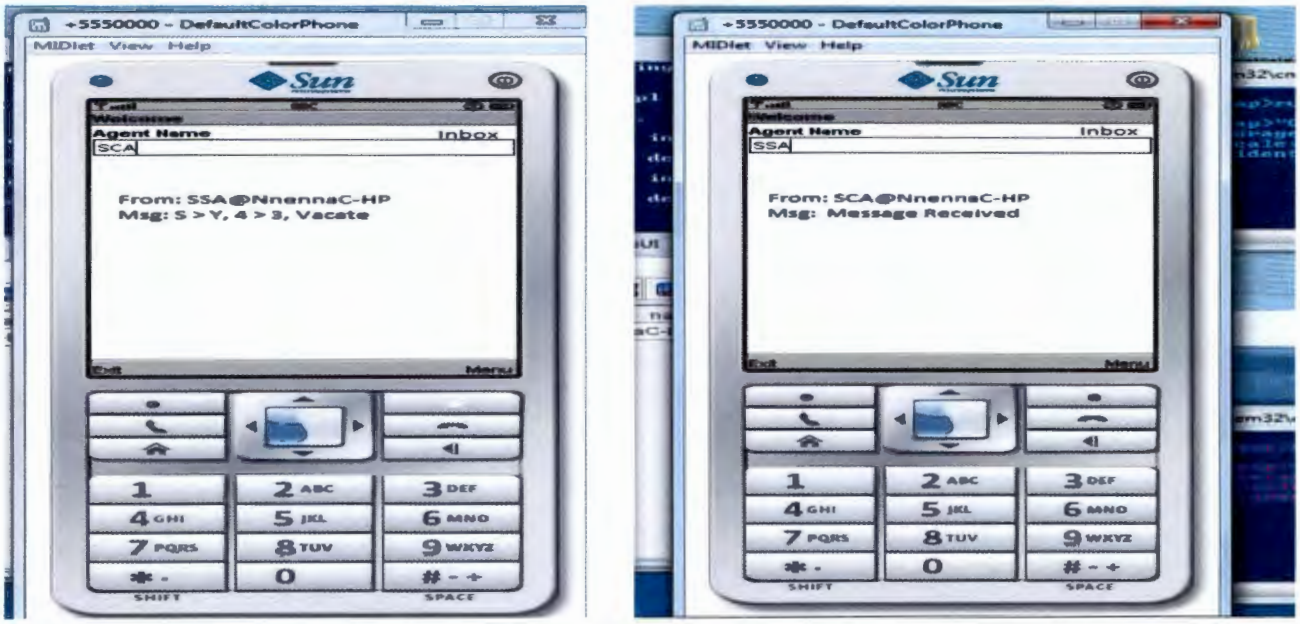


Fig. 4.11: Spectrum Mobility Message

The experiment for spectrum mobility was conducted a number of times to determine the switching time, which is the difference between the time of primary user reappearance and the vacating time of the secondary user currently utilizing the channel. Table 4.5 shows the result obtained at different times of the reappearance of the primary user.

Table 4.5: Spectrum Mobility Times

Time of PU Reappearance	Time of SU Vacating	Switching Time (seconds)
12:25:04:08	12:25:04:13	0.5
12:45:32:20	12:45:32:28	0.8
1:16:02:00	1:16:02:02	0.2
1:30:15:24	1:30:15:30	0.6
2:05:14:56	2:05:15:00	0.4

The results in Table 4.5 shows that the switching times were quite small, therefore causing no or minimal interference to other users. This was made possible because the radios had a foreknowledge of the reappearance of the primary user, based on messages sent by the agents.

4.4 Chapter Summary

This chapter presented the implementation of IMACRN agents in Java Agent Development Environment (JADE). The agents created were able to search for spectrum, decide on the appropriate channel using the directory facility, acquire spectrum from primary user agents, share spectrum amongst secondary users and send messages to other agents and radios in the case of spectrum handoff. These results obtained, though not 100%, clearly show that our system, IMACRN, can be used to efficiently manage spectrum resource in cognitive radio networks.

Chapter 5

Summary, Conclusion and Future Work

5.1 Summary

A driving feature of future network architectures will be the mobile user. Users increasingly will access information resources while on the move, whether when in a vehicle, etc. Wireless technology is necessary to support the mobile user and adaptive and efficient use of radio spectrum is an important aspect of developing future network architectures. Observing that in some locations or at some times of day, 70% of the allocated spectrum may be sitting idle, the FCC has recently recommended that significantly greater spectral efficiency could be realized by deploying wireless devices that can coexist with the primary users, generating minimal interference while taking advantage of the available resources. Thus, the discrepancy between spectrum allocation and spectrum use suggests that this spectrum shortage could be overcome by allowing more flexible usage of a spectrum. Flexibility would mean that radios could find and adapt to any immediate local spectrum availability. A new class of radios that is able to reliably sense the spectral environment over a wide bandwidth, detects the presence or absence of users, and use the spectrum only if the communication does not interfere with primary users, is defined by the term cognitive radio.

Cognitive Radios integrate radio technology and networking technology to provide efficient use of radio spectrum. The cognitive radio wireless network is intended as an advanced technology integration environment with focus on building adaptive, spectrum-efficient systems with emerging programmable radios. The emerging cognitive radio scenario is of current interest to both policy makers and technologists because of the potential for order of magnitude gains in spectral efficiency and network performance. The idea of a cognitive radio extends the concepts of a hardware radio and a software defined radio from a simple, single function device to a radio that senses and reacts to its operating environment. The promise of cognitive radios is improved use of spectrum resources, reduced engineering and planning time, and adaptation to current operating conditions.

5.2 Conclusion

The rapid proliferation of wireless technologies is expected to increase the demand for radio spectrum by orders of magnitude over the next decade. This problem must be addressed via technology and regulatory innovations for significant improvements in spectrum efficiency and increased robustness/performance of wireless devices. Emerging cognitive radio technology has been identified as a high impact disruptive technology innovation, which could provide solutions to the "radio traffic jam" problem and provide a path to scaling wireless systems for the next 25 years.

Cognitive radio network represent a paradigm shift in both radio and networking technologies, with the potential to provide major gains in performance and spectrum efficiency. However, even as cognitive radio platforms have started to emerge, significant new research work is required to address the many technical challenges of cognitive radio networking. These include dynamic spectrum allocation (DSA) methods, spectrum sensing, cooperative communications, incentive mechanisms, cognitive network architecture and protocol design, cognitive network security, cognitive system adaptation algorithms and emergent system behaviour.

Conclusively, this research has been able to apply intelligent mobile agent to (a) detect idle spectrum, (b) decide if it is the best channel to use, (c) share the spectrum fairly amongst users in the network and (d) vacate the channel when the spectrum owner reappears. These functionalities, collectively called spectrum management, ensure efficient use of the spectrum as well as non-interference to the license owner.

5.3 Future Work

Because research in cognitive radio is practically new, there exist open issues that must be addressed. Some of the future directions as an extension for this research are in intelligent interference-aware and situation-aware mechanisms for cognitive radio networks. Another improvement can be designing algorithms that will consider the impact of in-band or out-band transmission channels for information exchange between cognitive radios with the help of agents.

References

[1] NSF Workshop Report. "Future Directions in Cognitive Radio Network (CRN) Research". NSF Workshop, Arlington, Virginia, March 9 – 10, 2009. pp 1 – 40.

[2] D.Cabric, S.M. Mishara and R.W. Brodersen. "Implementation Issues in Spectrum Sensing", *in Proc. 38th Annual Asilomar Conference on Signal, Systems and Computers.* Pacific Grove, CA, USA, November 2004. pp. 772 – 776.

[3] J. Ma, G. Y Li. "Signal Processing in Cognitive Radio", *in proc. of the IEEE*, Vol. 97, No 5, May 2009. pp. 805 – 823

[4] I.F. Akyildiz, W.Y. Lee, M.C. Vuran and S. Mohanty, "A Survey on Spectrum Management in Cognitive Radio Networks", *IEEE Communications Magazine*, Vol. 46, pp. 40-48, April 2008.

[5] I.F. Akyildiz, W.Y. Lee, M.C. Vuran and S. Mohanty, "Next Generation / Dynamic Spectrum Access / Cognitive Radio Networks: A Survey". *Computer Networks Journal*, Vol. 50, pp. 2127-2159, September 2006.

[6] J.S.K. Wong and A.R. Mikler, "Intelligent Mobile Agents in Large Distributed Autonomous Cooperative System". *The Journal of Systems and Software*, Issue 47, 1999, pp. 75-87

[7] M. Li and R.S.H. Istepanian, "3G Network Oriented Mobile Agents for Intelligent Diabetes Management: A Conceptual Model", *in proc. of The 4th Annual IEEE Conference on Information Technology Applications in Biomedicine*, UK, 2003, pp. 31-34

[8] Federal Communication Commission (FCC). "Notice of Proposed Rule Making and Order". ET Docket No 03 – 322, December 2003.

[9] Federal Communication Commission (FCC). "Spectrum Policy Task Force Report". Report ET Docket No 02–135, November 2002.

[10] B. Joy, G. Steele, J. Gosling and G. Bracha. "*The Java Language Specification*". 2nd ed. The Java Series. Addison-Wesley, June 2000, pp. 56-71.

[11] P. Hansen. " Java's insecure parallelism". SIGPLAN Notices 34, April 1999, pp. 38–45.

[12] Institute of Electrical and Electronics Engineers, Inc. "Information Technology: Portable Operating System Interface (POSIX),Part 1: System Application Program Interface (API) [C Language]". Std 1003.1, pp. 48-56, 1996.

[13] G. Czajkowski, L.Dayn_es and M. Wolczko. "Automated and portable native code isolation". Technical Report, Sun Microsystems Laboratories, pp. 01-96, April 2001.

[14] S. Liang. "*The Java Native Interface: Programmer's Guide and Specification*", 1st ed. The Java Series. Addison-Wesley, June 1999.

[15] S. Liang and G. Bracha. "Dynamic class loading in the Java virtual machine", *in Proc. of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '98)*, Vancouver, BC, October 1998, pp. 36–44.

[16] P. Bothner. "Kawa: compiling dynamic languages to the Java VM", *in Proc. of the USENIX 1998 Technical Conference, FREENIX Track*, New Orleans, June 1998, pp. 261–272.

[17] U. Mir, L. Merghem-Boulahia, M. Esseghir and D. Gaïti. "Dynamic Spectrum Sharing for Cognitive Radio Networks using Multi-agent System," *in proc. of the 8th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2011, pp. 203-215.

[18] U. Mir, L. Merghem-Boulahia and D. Gaïti. "A cooperative multi-agent based spectrum sharing," *in Proc. of the 6th Annual Advanced International Conference on Telecommunications*, Barcelona, 2010, pp. 150-165.

[19] X. Jiang, H. Ivan and R. Anita."Cognitive radio resource management using multi-agent systems," *in Proc. of the 4th Annual IEEE International Conference on Consumer Communications and Networking*, Vegas, 2007, pp. 1123-27.

[20] C. Zhang, V. Lesser and P. Shenoy. "A multi-agent learning approach to online distributed resource allocation," *in Proc. of the International Joint Conference on Artificial Intelligence (IJCAI 09)*, 2009, pp. 45-53.

[21] Z. Feng, K. Yu, Y. Ji, P. Zhang, L. Vok and Y. Zhang. "Multi-access Radio Resource Management using Multi-Agent System", *in proc. of IEEE Wireless Communications and Networking Conference*, Las Vegas, Nevada, USA, 3-6 April 2006, volume 1, p. 63-68.

[22] H. M. Kelash, H. M. Faheem and M. Amoon. "A Multi-agent System for Distributed Systems Management", *in Proc. of world academy of science, engineering and technology conference*, ISSN 1307-6884, volume 11, February 2006, pp. 91-96.

[23] J. Luo, R. Mukerjee and M. Dillinger. "Investigation of radio resource scheduling in WLANs coupled with 3G cellular network," *IEEE Communications Magazine*, 41(6): pp. 108 – 115, 2003.

[24] J. Luo, E. Mohyeldin, M. Dillinger, P. Demestichas, K. Tsagkaris, G. Dimitrakopoulos and E. Schulz. "Performance Analysis of Joint Radio Resource Management for Reconfigurable Terminals with Multi-class Circuit-switched Services," *WWRF 12th Meeting, WG6*, Toronto, Canada, November 2004, pp. 15-56.

[25] P. Magnusson, J. Lundsjo, J. Sachs and P. Wallentin, "Radio resource management distribution in a beyond 3G multi-radio access architecture," *in proc. of IEE Global Telecommunications Conference*, 29 Nov.- 3 Dec. 2004, vol. 6, pp. 3472 – 3477.

[26] S. Haykin. "Cognitive radio: brain-empowered wireless communications". *IEEE Journal of Selected Areas in Communication*, vol.23, no. 2, pp. 201-220, February 2005.

[27] I. F. Akyildiz, W. Lee, M. C. Vuran, and S. Mohanty. "NeXt generation/dynamic spectrum access/cognitive radio wireless networks: A survey," *Computer Networks Journal*, vol. 50, no. 13, pp. 2127- 2159, September 2006.

[28] IEEE 802.11 WG draft supplement: "Specification for radio resource measurement", IEEE 802.11k/D0.7.

[29] D. Harrington, R. Presuhn and B. Wijnen. "An architecture for describing simple network management protocol (SNMP) management frameworks," Request for Comments (RFC) 3411, IETF, December 2002.

[30] D. Culler, D. Estrin and M. Srivastava. "Overview of Sensor Networks", *IEEE Computer*, vol. 37, no. 8, pp. 41-49, August 2004.

[31] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister. "System Architecture directions for Networked Sensors", *in proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 93-104, 2000.

[32] P. Levis and D. Culler. "Mate: A tiny virtual machine for sensor networks", *in proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93-104.

[33] J. Zhao and D. Culler."Understanding packet delivery performance in dense wireless sensor networks", *in Proc. of the ACM SenSym*, 2003, pp. 48-56.

[34] D. Georgoulas and K. Blow. "In-Motes: An Intelligent Agent Based Middleware for Wireless Sensor Networks". Best Student Paper Award, *in Proc. of the 5th WSEAS International Conference SEPADS 06*, 2006, pp. 225-231.

[35] C-L Fok, G-C Roman and C. Lu. "Rapid development and flexible deployment of adaptive wireless sensor networks", *inProc. of the 24th International Conference ICDS 05*, 2005, pp.37-47.

[36] M. Genesereth and S. Ketckpel. "Software Agents", *in Communications of ACM*, pp.48-53, 1994.

[37] C. Roadknight and I. Marshall. "Future Network Management: A bacterium inspired solution", *in Proc. of the 2nd International Symposium on Engineering and Intelligent Systems*, 2000, pp. 87-96.

[38] D. Georgoulas and K. Blow. "Making Motes Intelligent: An agent based approach to wireless sensor networks", *WSEAS Communications Journal*, pp 515-522, 2006.

[39] D. Georgoulas and K. Blow. "Intelligent Mobile Agent Middleware for Wireless Sensor Networks: A Real Time Application Case Study", *IEEE Computer*, pp. 35-48, March 2007.

[40] C. Wu, K. Chowdhury, M. D. Felice and W. Meleis. "Spectrum Management of Cognitive Radio Using Multi-Agent Reinforcement Learning", *in proc. of the 9th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2010)*, Toronto, Canada, May 10-14, 2010, pp. 1705-1712.

[41] A. Ahmed, M. M. Hassan, O. Sohaib, W. Hussain and M. Q. Khan. "An Agent Based Architecture for Cognitive Spectrum Management". *Australian Journal of Basic and Applied Sciences*, vol. 5, no. 12, pp. 682-689, 2011.

[42] F. Mark. "Non Cooperative Multi-Radio Channel Allocations in Wireless Networks". *Infocom, IEEE*, pp. 1442-1450, 2007.

[43] F. L.Bellifemine, G. Caire and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Hoboken, NJ: Wiley, 2007, pp. 12-46.

[44] FIPA. "Agent Management Support for Mobility Specifications". Foundation for Intelligent Physical Agents. Geneva, Switzerland, pp. 10-18, 2000.

[45] FIPA. "Abstract Architectural Specifications". Foundation for Intelligent Physical Agents. Geneva, Switzerland, pp. 25-38, 2002.

[46] C. Kloeck, H. Jaekel and F. Jondral. "Multi-Agent Resource Allocation". *Mobile Networks and Applications Journal*, vol. 11, no. 6, pp. 813-824, 2006.

APPENDIX

```
import java.util.*;
import java.io.*;
import jade.lang.acl.*;
import jade.content.*;
import jade.content.onto.basic.*;
import jade.content.lang.*;
import jade.content.lang.sl.*;
import jade.core.*;
import jade.core.behaviours.*;
import jade.domain.*;
import jade.domain.mobility.*;
import jade.domain.JADEAgentManagement.*;
import jade.gui.*;


public class ControllerAgent extends GuiAgent {
// ----------------------------------------


  private jade.wrapper.AgentContainer home;
  private jade.wrapper.AgentContainer[] container = null;
  private Map locations = new HashMap();
  private Vector agents = new Vector();
  private int agentCnt = 0;
  private int command;
  transient protected ControllerAgentGui myGui;
```

```
  public static final int QUIT = 0;
  public static final int NEW_AGENT = 1;
  public static final int MOVE_AGENT = 2;
  public static final int CLONE_AGENT = 3;
  public static final int KILL_AGENT = 4;

  // Get a JADE Runtime instance
  jade.core.Runtime runtime = jade.core.Runtime.instance();

  protected void setup() {
// ------------------------

        // Register language and ontology
        getContentManager().registerLanguage(new SLCodec());
        getContentManager().registerOntology(MobilityOntology.getInstance());

    try {
      // Create the container objects
      home = runtime.createAgentContainer(new ProfileImpl());
      container = new jade.wrapper.AgentContainer[3];
      for (int i = 0; i < 5; i++){
        container[0] = runtime.createAgentContainer(new ProfileImpl());
            }
            doWait(2000);

            // Get available locations with AMS
            sendRequest(new Action(getAMS(), new QueryPlatformLocationsAction()));
```

```
        //Receive response from AMS
    MessageTemplate mt = MessageTemplate.and(
                                    MessageTemplate.MatchSender(getAMS()),
                                    MessageTemplate.MatchPerformative(ACLMessage.INFORM));
    ACLMessage resp = blockingReceive(mt);
    ContentElement ce = getContentManager().extractContent(resp);
    Result result = (Result) ce;
    jade.util.leap.Iterator it = result.getItems().iterator();
    while (it.hasNext()) {
      Location loc = (Location)it.next();
      locations.put(loc.getName(), loc);
                }
        }
        catch (Exception e) { e.printStackTrace(); }


        // Create and show the gui
   myGui = new ControllerAgentGui(this, locations.keySet());
   myGui.setVisible(true);
  }


  protected void onGuiEvent(GuiEvent ev) {
// ------------------------------------


        command = ev.getType();
```

```
        if (command == QUIT) {
          try {
                    home.kill();
                    for (int i = 0; i < container.length; i++) container[i].kill();
          }
          catch (Exception e) { e.printStackTrace(); }
          myGui.setVisible(false);
          myGui.dispose();
                doDelete();
                System.exit(0);
}
        if (command == NEW_AGENT) {

          jade.wrapper.AgentController a = null;
    try {
      Object[] args = new Object[2];
      args[0] = getAID();
      String name = "Agent"+agentCnt++;
      a = home.createNewAgent(name, MobileAgent.class.getName(), args);
            a.start();
            agents.add(name);
            myGui.updateList(agents);
          }
    catch (Exception ex) {
                    System.out.println("Problem creating new agent");
          }
    return;
```

```
        }
String agentName = (String)ev.getParameter(0);
AID aid = new AID(agentName, AID.ISLOCALNAME);

        if (command == MOVE_AGENT) {

  String destName = (String)ev.getParameter(1);
  Location dest = (Location)locations.get(destName);
  MobileAgentDescription mad = new MobileAgentDescription();
  mad.setName(aid);
  mad.setDestination(dest);
  MoveAction ma = new MoveAction();
  ma.setMobileAgentDescription(mad);
  sendRequest(new Action(aid, ma));
        }
else if (command == CLONE_AGENT) {

  String destName = (String)ev.getParameter(1);
  Location dest = (Location)locations.get(destName);
  MobileAgentDescription mad = new MobileAgentDescription();
  mad.setName(aid);
  mad.setDestination(dest);
  String newName = "Clone-"+agentName;
  CloneAction ca = new CloneAction();
  ca.setNewName(newName);
  ca.setMobileAgentDescription(mad);
  sendRequest(new Action(aid, ca));
```

```
      agents.add(newName);
      myGui.updateList(agents);
          }
    else if (command == KILL_AGENT) {
      KillAgent ka = new KillAgent();
      ka.setAgent(aid);
      sendRequest(new Action(aid, ka));
            agents.remove(agentName);
                  myGui.updateList(agents);
          }
  }
  void sendRequest(Action action) {
// --------------------------------
    ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
    request.setLanguage(new SLCodec().getName());
    request.setOntology(MobilityOntology.getInstance().getName());
    try {
            getContentManager().fillContent(request, action);
            request.addReceiver(action.getActor());
            send(request);
          }
          catch (Exception ex) { ex.printStackTrace(); }
  }

}//class Controller
```

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;
import javax.swing.border.*;

import jade.core.*;
import jade.gui.*;


public class ControllerAgentGui extends JFrame implements ActionListener {
// ------------------------------------------------------------------

  private JList list;
  private DefaultListModel listModel;
  private JComboBox locations;
  private JButton newAgent, move, clone, kill, quit;
  private ControllerAgent myAgent;

  public ControllerAgentGui(ControllerAgent a, Set s) {
// ------------------------------------------------------

    super("Controller");
    this.myAgent = a;
    JPanel base = new JPanel();
```

```
base.setBorder(new EmptyBorder(15,15,15,15));
base.setLayout(new BorderLayout(10,0));
        getContentPane().add(base);



        JPanel pane = new JPanel();
        base.add(pane, BorderLayout.WEST);
pane.setLayout(new BorderLayout(0,10));
listModel = new DefaultListModel();
list = new JList(listModel);
list.setBorder(new EmptyBorder(2,2,2,2));
list.setVisibleRowCount(5);
list.setFixedCellHeight(18);
list.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
pane.add(new JScrollPane(list), BorderLayout.NORTH);

JPanel p = new JPanel();
p.setLayout(new GridLayout(1,2,5,0));
p.add(new JLabel("Destination :"));
locations = new JComboBox(s.toArray());
p.add(locations);
pane.add(p, BorderLayout.CENTER);

p = new JPanel();
p.setLayout(new GridLayout(1,3,5,0));
p.add(move = new JButton("Move"));
move.setToolTipText("Move agent to a new location");
```

```
move.addActionListener(this);
p.add(clone = new JButton("Clone"));
clone.setToolTipText("Clone selected agent");
clone.addActionListener(this);
p.add(kill = new JButton("Kill"));
kill.setToolTipText("Kill selected agent");
kill.addActionListener(this);
pane.add(p, BorderLayout.SOUTH);
        move.setEnabled(false);
        clone.setEnabled(false);
        kill.setEnabled(false);
list.addListSelectionListener( new ListSelectionListener() {
                public void valueChanged(ListSelectionEvent e) {
                        if (list.getSelectedIndex() == -1){
                          move.setEnabled(false);
                          clone.setEnabled(false);
                          kill.setEnabled(false);
                        }
                        else {
                          move.setEnabled(true);
                          clone.setEnabled(true);
                          kill.setEnabled(true);
                        }
                }
        });
pane = new JPanel();
pane.setBorder(new EmptyBorder(0,0,110,0));
```

```
        base.add(pane, BorderLayout.EAST);
    pane.setLayout(new GridLayout(2,1,0,5));
    pane.add(newAgent = new JButton("New agent"));
    newAgent.setToolTipText("Create a new agent");
    newAgent.addActionListener(this);
    pane.add(quit = new JButton("Quit"));
    quit.setToolTipText("Terminate this program");
    quit.addActionListener(this);

    addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                shutDown();
            }
        });

    setSize(300, 210);
    setResizable(false);
    pack();
  }

  public void actionPerformed(ActionEvent ae) {
// ------------------------------------

    if (ae.getSource() == newAgent) {

      GuiEvent ge = new GuiEvent(this, myAgent.NEW_AGENT);
      myAgent.postGuiEvent(ge);
```

```
        }
    else if(ae.getSource() == move) {

      GuiEvent ge = new GuiEvent(this, myAgent.MOVE_AGENT);
      ge.addParameter((String)list.getSelectedValue());
      ge.addParameter((String)locations.getSelectedItem());
      myAgent.postGuiEvent(ge);
        }
    else if (ae.getSource() == clone) {

      GuiEvent ge = new GuiEvent(this, myAgent.CLONE_AGENT);
      ge.addParameter((String)list.getSelectedValue());
      ge.addParameter((String)locations.getSelectedItem());
      myAgent.postGuiEvent(ge);
        }
        else if (ae.getSource() == kill) {

      GuiEvent ge = new GuiEvent(this, myAgent.KILL_AGENT);
      ge.addParameter((String)list.getSelectedValue());
      myAgent.postGuiEvent(ge);
        }
    else if (ae.getSource() == quit) {
      shutDown();
        }
}

void shutDown() {
```

```
// ---------- Control the closing of this gui

    GuiEvent ge = new GuiEvent(this, myAgent.QUIT);
    myAgent.postGuiEvent(ge);
  }

  public void updateList(Vector v) {
// -------------------------------

    listModel.clear();
    for (int i = 0; i < v.size(); i++){
      listModel.addElement(v.get(i));
          }
  }

}
```

```
import java.util.*;
import java.io.*;

import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.content.*;
import jade.content.lang.*;
import jade.content.lang.sl.*;
import jade.content.onto.basic.*;
import jade.domain.*;
import jade.domain.mobility.*;
import jade.domain.JADEAgentManagement.*;
import jade.gui.*;


public class MobileAgent extends GuiAgent {
// ------------------------------------

  private AID controller;
  private Location destination;
  transient protected MobileAgentGui myGui;

  protected void setup() {
// ----------------------
```

```
        // Retrieve arguments passed during this agent creation
        Object[] args = getArguments();
        controller = (AID) args[0];
        destination = here();

        init();

        // Program the main behaviour of this agent
    addBehaviour(new ReceiveCommands(this));
  }

  void init() {
// ------------

        // Register language and ontology
        getContentManager().registerLanguage(new SLCodec());
        getContentManager().registerOntology(MobilityOntology.getInstance());

        // Create and display the gui
        myGui = new MobileAgentGui(this);
        myGui.setVisible(true);
        myGui.setLocation(destination.getName());
  }

  protected void onGuiEvent(GuiEvent e) {
// ----------------------------------
   //No interaction with the gui
```

```
  }

  protected void beforeMove() {
// ----------------------------

    System.out.println("Moving now to location : " + destination.getName());
    myGui.setVisible(false);
    myGui.dispose();
  }

  protected void afterMove() {
// ----------------------------

        init();
        myGui.setInfo("Arrived at location : " + destination.getName());
  }

  protected void beforeClone() {
// ----------------------------

    myGui.setInfo("Cloning myself to location : " + destination.getName());
  }

  protected void afterClone() {
// ----------------------------

        init();
```

```
  }


  /*
  * Receive all commands from the controller agent
  */
  class ReceiveCommands extends CyclicBehaviour {
// ----------------------------------------


    ReceiveCommands(Agent a) { super(a); }


          public void action() {


      ACLMessage msg = receive(MessageTemplate.MatchSender(controller));


      if (msg == null) { block(); return; }


            if (msg.getPerformative() == ACLMessage.REQUEST){


                  try {
                        ContentElement content = getContentManager().extractContent(msg);
                        Concept concept = ((Action)content).getAction();


                        if (concept instanceof CloneAction){


                              CloneAction ca = (CloneAction)concept;
                              String newName = ca.getNewName();
```

```
                    Location l = ca.getMobileAgentDescription().getDestination();
                    if (l != null) destination = l;
                    doClone(destination, newName);
                }
                else if (concept instanceof MoveAction){

                    MoveAction ma = (MoveAction)concept;
                    Location l = ma.getMobileAgentDescription().getDestination();
                    if (l != null) doMove(destination = l);
                }
                else if (concept instanceof KillAgent){

                    myGui.setVisible(false);
                    myGui.dispose();
                    doDelete();
                }
              }
              catch (Exception ex) { ex.printStackTrace(); }
          }
          else { System.out.println("Unexpected msg from controller agent"); }
      }
  }

} // class MobileAgent
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

import jade.gui.*;



public class MobileAgentGui extends JFrame {
// ------------------------------------

  private JTextField location;
  private JTextField info;
  private MobileAgent myAgent;

  public MobileAgentGui(MobileAgent a) {
// ------------------------------------

        myAgent = a;
    setTitle(myAgent.getLocalName());

        // Add button and text field
    Container c = getContentPane();
    JPanel base = new JPanel();
    base.setBorder(new EmptyBorder(20,20,20,20));
    getContentPane().add(base);
```

```
        base.setLayout(new BorderLayout(0,20));
        JPanel pane = new JPanel();
        pane.setLayout(new BorderLayout(10,0));
        pane.add(new JLabel("Current location : "), BorderLayout.WEST);
        pane.add(location = new JTextField(15), BorderLayout.EAST);
        location.setEditable(false);
        location.setBackground(Color.white);
        base.add(pane, BorderLayout.NORTH);
        base.add(info = new JTextField(20), BorderLayout.SOUTH);
        info.setEditable(false);
        info.setHorizontalAlignment(JTextField.CENTER);

        setSize(200,100);
        pack();
        setResizable(false);
        Rectangle r = getGraphicsConfiguration().getBounds();
        setLocation(r.x + r.width-getWidth(), r.y);
}

public void setLocation(String loc){

  this.location.setText(loc);
}
```