Impact of Aspect-Orientation on the Reliability of Decentralized Multi-Agent System

UNDERTAKEN BY:

HIRA TABBASUM

36-CS/MSSE/2003

SALMA JABEEN

37-CS/MSSE/2003

SUPERVISED BY:

DR. H. FAROOQ AHMAD

Faculty of Basic & Applied Sciences

International Islamic University Islamabad

A dissertation submitted to the

Department of Software Engineering,

Faculty of Applied Sciences,

International Islamic University, Islamabad, Pakistan,

as a partial fulfillment of the requirements for the award of the degree of

MS in Software Engineering

To

The Holiest Man Ever Born,

PROPHET MUHAMMAD (صلى الله عليه وسلم)

To

OUR PARENTS AND FAMILIES

We are most indebted to our parents and families, whose affection has always been the source of encouragement for us, and whose prayers have always been a key to our success.

To

THOSE HOLY SEEKERS

Who give away their lives to make the stream of life flow Smoothly and with Justice.

And

To

OUR HONORABLE TEACHERS

Who have been a beacon of knowledge and a constant source of inspiration, for our whole life span.

Declaration

We hereby declare and affirm that this software neither as a whole nor as a part thereof has been copied out from any source. It is further declared that we have developed this software and accompanied report entirely on the basis of our personal efforts, made under the sincere guidance of our teachers. If any part of this project is proven to be copied out or found to be a reproduction of some other, we shall stand by the consequences.

No portion of the work presented in this report has been submitted in support of an application for other degree or qualification of this or any other University or Institute of learning.

Hira Tabbasum
36-CS/MSSE/03
Salma Jabeen
37-CS/MSSE/03

Acknowledgement

We bestow all praises, acclamation and appreciation to Almighty Allah, The Most Merciful and Compassionate, The Most Gracious and Beneficent, Whose bounteous blessings enabled us to pursue and perceive higher ideas of life, Who bestowed us good health, courage, and knowledge to carry out and complete our work. Special thanks to our Holy Prophet Muhammad (SAW) who enabled us to recognize our Lord and Creator and brought us the real source of knowledge from Allah (SWT), the Qur'ān, and who is the role model for us in every aspect of life.

We consider a proud privilege to express our deepest gratitude and deep sense obligation to our reverend supervisor **Dr. H. Farooq Ahmad** who kept our morale high by his suggestions and appreciation. His motivation led us to this success. Without his sincere and cooperative nature and precise guidance; we could never have been able to complete this task.

It will not be out of place to express our profound admiration and gratitude for our teacher **Dr. Naveed Ikram** for his dedication, inspiring attitude and kind behavior throughout the project efforts and presentation of this manuscript.

Finally we must mention that it was mainly due to our parent's moral support and financial help during our entire academic career that enabled us to complete our work dedicatedly. We owe all our achievements to our most loving parents, who mean most to us, for their prayers are more precious before any treasure on earth. We are also thankful to our loving brothers, sisters, friends, and class fellows who mean the most to us, and whose prayers have always been a source of determination for us.

Hira Tabbasum
36-CS/MSSE/03
Salma Jabeen
37-CS/MSSE/03

Abstract

Aspect Oriented Programming (AOP) provides separation of concerns and encapsulates crosscutting concerns into separate modules called 'aspects', thereby enhances the software quality. This thesis presents a quantitative study that assesses the positive and negative impacts of AOP on the reliability of a decentralized Multi-Agent System. Comparison between aspect-oriented and object-oriented versions of the same application, SAGE (scalable and fault tolerant agent grooming environment), is performed, in order to explore to what extent each implementation provides a reliable system. Reliability depends upon the internal attributes like coupling and cohesion and is inversely proportional to complexity. Refinement of Chidamber and Kemerer metrics suite is used to evaluate aspect-oriented version and calculated coupling, cohesion and complexity, those are the basic error-prone in a system and afterwards Mean Time to Failure is measured for both versions of the system. It is found that aspect-oriented system which showed good results for Chidamber and Kemerer metrics suite also showed good results for Mean Time to Failure.

PROJECT IN BRIEF

Project Title:	**Impact of Aspect Orientation on the Reliability of Decentralized Multi-Agent System**
Organization:	International Islamic University, Islamabad
Undertaken By:	**Hira Tabbasum** **Reg. No: 36-CS/MSSE/03** **Salma Jabeen** **Reg. No: 37-CS/MSSE/03**
Supervised By:	**Dr. Hafiz Farooq Ahmad** **Associate Professor** **Nust Institute of Information Technology, Rawalpindi**
Starting Date:	October, 2007
End Date:	August, 2008
Tools Used:	Eclipse SDK (3.1.0), Rational Rose 2002, AspectJ 1.5
System Used:	Pentium IV

Table of Contents

Table of Figures

Table of Tables

Chapter 1

Introduction

Chapter 1: Introduction

1.1 Introduction

Aspect Oriented Software Development (AOSD) is a new emerging technology that provides separation of concerns in software construction [1]. Separation of concerns is a central software engineering principle that should be applied throughout the development process, from requirement to implementation [2]. It states that a given problem involves different kinds of concerns, which should be identified and separated in order to manage the complexity of the system [3]. Concerns can be classified into two categories those are core concerns and crosscutting concerns. Core concerns deal with the basic functionality of a system while crosscutting concerns span multiple modules and deal with non-functional requirements [4]. Aspect Oriented Programming provides improved modularization which encapsulates crosscutting concerns into separate modules known as 'aspects' [5].

Software engineering of multi-agent system involves the classification of concerns into two categories: agenthood concerns and additional concerns. Agenthood concerns include knowledge, interaction, adaptation, and autonomy. While additional concerns include mobility, learning and collaboration. Out of these mobility, interaction, learning, autonomy and collaboration are crosscutting concerns [6]. A group of researchers have worked on aspect-oriented architecture [6], modeling [7] and engineering [1] of multi-agent systems but there is no empirical evidence whether AOP helps in improving the reliability of a decentralized Multi-Agent system, thereby hindering the adaptation of AOP for such system.

This thesis presents a case-study in which we have compared the reliability of aspect-oriented (AO) and object-oriented (OO) design of decentralized Agent Management System (AMS) of SAGE (scalable, fault tolerant agent grooming environment). SAGE is FIPA [8] compliant decentralized multi-agent system [9]. In SAGE, scalability and fault tolerance is achieved up to some extent through its architecture but reliability is still a major issue due to the excess of tangling and scattering of code in one of its components i.e., AMS. In addition, internal attributes like coupling and cohesion also affect system's external attributes like reliability, reusability and maintainability [10] while SAGE is a highly coupled and complex system because its code is not well optimized [11].

This research also explains crosscutting concerns those come across the development of a decentralized AMS and implemented those concerns with AspectJ [12]. It involves three sorts of crosscutting concerns, knowledge distribution, exception handling and knowledge consistency and core concerns, such as agent management and peer management. We also evaluated both the versions of decentralized AMS through metrics for reliability named Mean Time to Failure and on the basis of the results which we got from evaluation, we compared both the systems.

1.2 Background Information

Since our research is based on convergence of two disciplines i.e., aspect-oriented paradigm and agent-oriented systems so first of all we define some terms used in this thesis and our research related to aspect-oriented programming and multi-agent system respectively.

1.2.1 Aspect-Oriented Programming

Aspect Oriented Programming introduces concern abstraction. It's a common accepted premise that the best way of dealing with complexity is to simplify it. In software design, the best way of simplifying a complex system is to identify the concerns and then modularize them. In fact, the OOP methodology was developed as a response to the need to modularize the concerns of a software system. The reality is, though, that although OOP is good at modularizing core concerns, it falls short when it comes to modularizing the crosscutting concerns. The AOP methodology was developed to address the shortfall. In AOP, the crosscutting concerns are modularized by identifying a clear role for each one in the system, implementing each role in its own module, and loosely coupling each module to a limited number of other modules [4]. The terminologies come under the Aspect-Oriented technology are as follows [13]:

Code tangling: If crosscutting concerns are implemented without being noticed, the code for concerns becomes intermixed i.e., code for crosscutting concerns finds itself scattered throughout multiple modules.

Aspects: A modular unit designed to implement a concern. It may contain some code and the instructions on where, when and how to invoke it.

Join points: Join points are well-defined places in the structure or execution flow of a program where additional behavior can be attached.

Advice: Advice is the behavior to execute at a join point. Many aspect languages provide mechanism to run advice before, after, instead of, or around join points of interest.

Weaving: It is the process of composing the core functionality modules with aspects, thereby yielding a working environment.

1.2.2 Multi-agent Systems (MAS)

Multi-agent systems [14] are systems composed of multiple agents, which interact with one another, typically by exchanging messages through some computer network infrastructure. MAS provide proper execution environment to agents so that they can assure the provision of services to other agents by cooperating, coordinating, and negotiating.

MAS represent virtual societies where software entities (agents) acting on behalf of their owners or controllers (people or organizations) can meet and interact for various reasons (e.g., exchanging goods, combining services, etc.) and in various ways (e.g., creating virtual organizations, participating to auctions, etc.)

1.2.3 Foundation for Intelligent Physical Agents (FIPA)

Foundation for Intelligent Physical Agents (FIPA) is a standard governing body for Agent development community. It provides abstract architecture of a complete Multi-agent System. Concrete realization of the abstract architecture will be according to the choice of the developer. Till now many FIPA compliant MAS have been implemented, JADE is one of the examples of FIPA compliant MAS [8].

1.2.4 Agent Management System

An Agent Management System (AMS) is a mandatory component of the Agent Platform (AP). The AMS exerts supervisory control over access to and use of the AP. Only one AMS will exist in a single AP. The AMS maintains a directory of AIDs (Agent ID's). Every Agent will have a unique Agent ID. AID will contain Transport Addresses (amongst other things) for agents to be registered with the AP [16].

1.2.4.1 General Architecture

Each agent must register with an AMS in order to get a valid AID. The AMS is responsible for managing the operation of an AP.

General Architecture

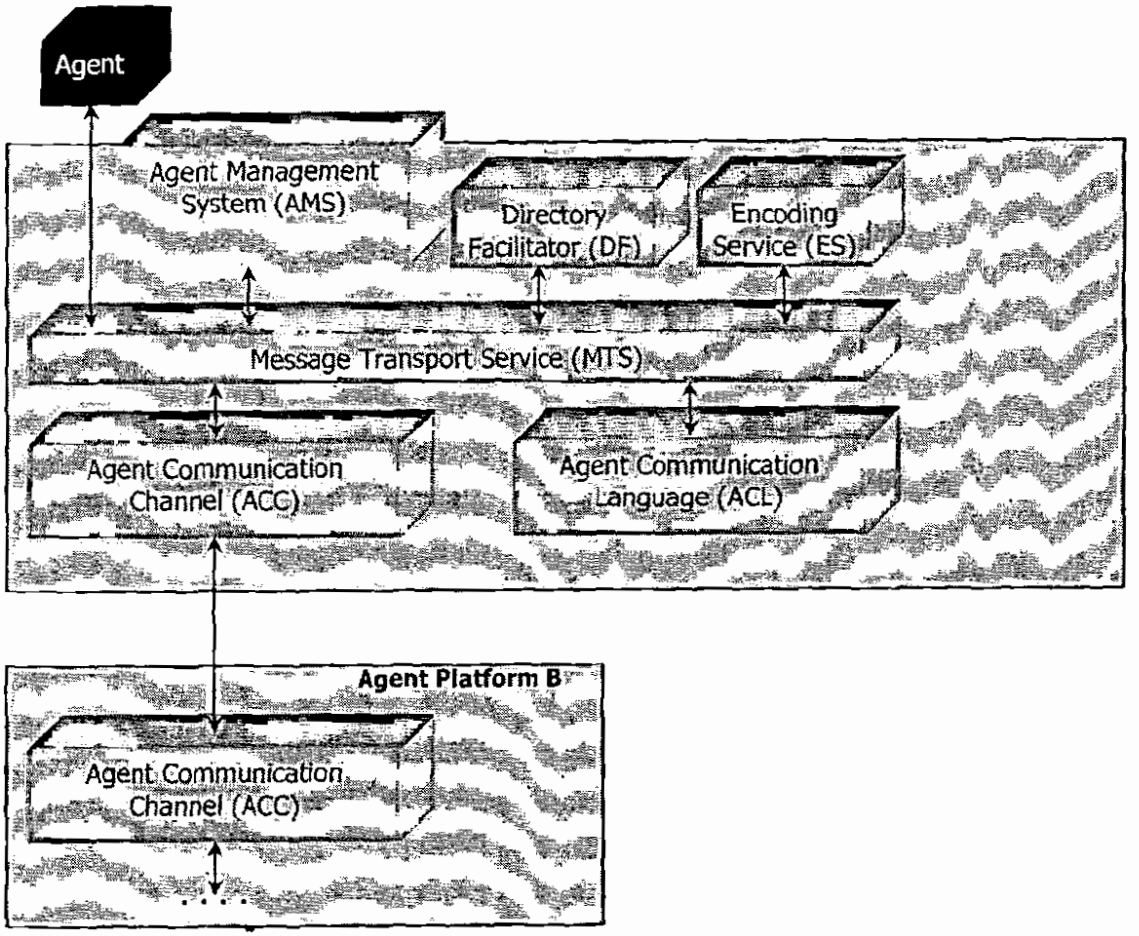

Fig 1.1: General Architecture of Multi Agent System

Figure 1.1 shows the general architecture of a multi-agent system [8].

The Agent Platform has following components:

- Agent Management System (AMS)
- Directory Facilitator (DF)
- Agent Communication Language (ACL)
- Message Transport Service (MTS)
- Encoding Service (ES)

1.2.5 Decentralized Agent Management System

The basic idea of our proposal is application of aspect oriented software development on a decentralized multi agent system i.e. SAGE (scalable, fault tolerant agent grooming environment). SAGE is a FIPA compliant decentralized MAS [9]. Different developers worked on it in parallel while using different coding standards and techniques which made it complex. It is strongly coupled and code is not optimized as well [11]. Distributed multi agent systems lack fault tolerance because of centralized registry and management. SAGE is following distributed architecture with decentralized management for fault tolerance of multi-agent systems. Its architecture is a blend of merits of client/server paradigm and peer-to-peer. Instead of having centralized location for management the owner-ship rights have been distributed to peer entities which are solely responsible for their roles and actions. These peer entities are part of single Agent Platform and are managed by Agent Management System (AMS). In this section we are going to describe the basic functionality of AMS.

1.2.5.1 AMS-functionality

The AMS represents the managing authority of an agent platform and if the agent platform spans multiple machines; then the AMS represents the authority across all machines. On a single agent platform only one AMS can exist.

Figure 1.2 shows the functionality of a typical AMS [16]. In SAGE, a typical AMS is responsible for sharing knowledge with other peer machines which is done with the help of peer manager. For instance, when a remote machine searches for a particular agent, peer manager's server checks the registry information providing agent name as the search criteria. If the agent is found the server responds by showing success otherwise it indicates failure.

The communication between peers is through AMS RMI layer and each RMI has a server and client on the same machine. A typical server application creates some remote objects (like rmiregistry), makes references to them accessible and waits for client to invoke methods on these remote objects. A client application gets a remote reference to one or more remote objects in the server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. For instance, when a remote machine joins the

platform, the client of the new machine calls the server of existing machine which on receiving request adds it by updating information in local cache. The information about the existing machines on the agent platform is given to the new machine and all the other machines are also provided the information about addition of new machine. So peer manager provides interaction among peers in this way. In addition, every peer probes other peer in order to check existence of other. Probing is done dynamically on every machine after a particular interval of time. A thread is created by agent directory service (ADS) that probes every machine after particular interval of time. In case of failure of remote machine the entry is removed from local as well as remote machine.

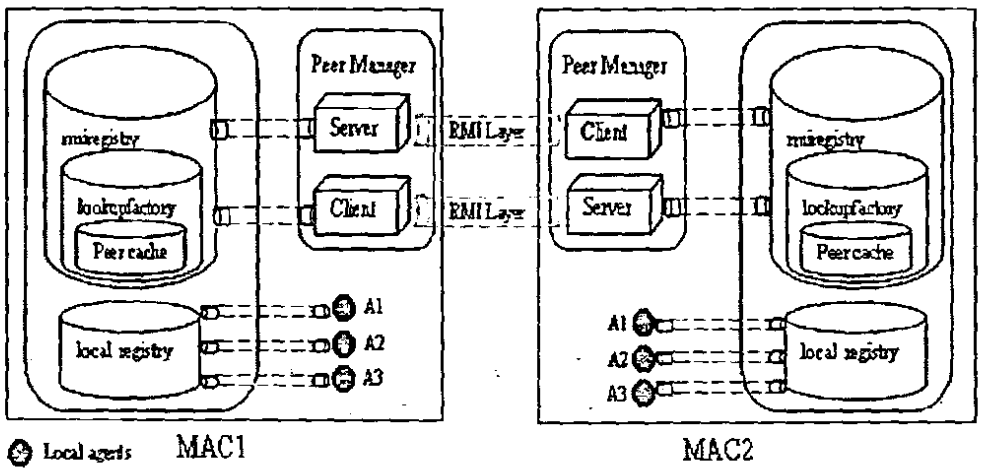

Fig 1.2: Functionality of AMS

Chapter 2

Literature Review

Chapter 2: Literature Review

2.1 Related Work

With the increase in size and complexity of multi-agent systems, the need for concern-separation in agent based applications, early in design phase also increases. Concerns like mobility, interaction, learning, autonomy and collaboration are crosscutting concerns in nature. Several authors [15, 17, 18, 19, 20, 21, 22] have emphasized on the separation of crosscutting and core concerns while designing a multi agent system. But so far no work has been done on the aspect-orientation of decentralized multi-agent systems. Some of the papers that write about aspect-oriented architecture, modeling and engineering of multi-agent systems are discussed as under:

Engineering Multi-Agent Systems with Aspects and Patterns [1]

When a complex problem can't be solved by a single agent and more agents are needed to work with, such systems are called Multi-Agent systems. A multi-agent system consists of multiple agents and objects. These are the abstractions used to model a specific problem. Objects and agents have some concerns in common but agents are more complex and have some additional concerns. Agents have two characteristics those are state and behavior. There state is determined by beliefs, goals, actions and plans while their behavior consists of some properties those include interaction, adaptation, learning, autonomy, mobility and collaboration. These concerns are not distinct in nature rather they have an overlapping relationship with one another and also have communication between them. This paper is based on a case-study of Portalware, a web-based environment, which constructs and manages ecommerce portals. This paper presents an Aspect-Oriented and Pattern based method for multi-agent system development and their comparison to show that which system is better in terms of understandability, maintainability and reusability.

Agents are of several types those are information agents, user agents and interface agents. Each agent type includes agency concerns. These concerns can be classified into three categories those are agent state, agency properties and agent role. Agent state is determined by knowledge which is based on beliefs, goals, actions and plans. Agency property and agent hood property forms behavior of an agent. Agenthood property includes autonomy, interaction and adaptation while learning, mobility and collaboration

are not necessary for agent-oriented systems. Agency properties are explained in Table 2.1 given below:

Agency Property	Definition
Interaction	An agent communicates with the environment and other agents by means of sensors and effectors
Adaptation	An agent adopts/modifies its mental state according to the messages received from the environment
Autonomy	An agent is capable of acting without direct intervention; it has its own control thread and can accept or refuse a request message
Learning	An agent can learn based on previous experience while reacting and interacting with its environment
Mobility	An agent is able to transport itself from one environment in a network to another
Collaboration	An agent can cooperate with other agents in order to achieve its goals and the system's goals

Table 2.1: An Overview of Agency Properties

Each agent application has a specific role for which it is developed. Agents co-operate and co-ordinate with one another to perform system's role. Agency properties like interaction, adaptation, learning, autonomy and collaboration have crosscutting nature. This paper applied Aspect-Oriented and Pattern Based Method for the development on a Portalware case study, after comparison concluded that aspect-oriented method enhances reusability, evolve-ability and writ-ability.

A Generative Approach for Multi-Agent System Development [23]

This paper says that multi-agent system development involves two types of concerns those are core and crosscutting. Many of the concerns involved in multi-agent system development are crosscutting in nature. Existing methodologies are too high level and do not deal with complexity. Implementation frameworks are not enough to deal with modeling and implementation of agent-oriented crosscutting concerns. This paper resolves the above mentioned issues by presenting a generative approach for development of multi-agent systems. This approach designs the agent-oriented system by using Agent-DSL language, then it makes aspect-oriented architecture and at the end a code generator generates the code for that multi-agent system. This paper also applied proposed approach on a case-study of ExpertCommitee system.

Aspectizing Multi-Agent Systems: From Architecture to Implementation [6]

Every agent architecture encompasses a set of agent properties: adaptation, autonomy, knowledge, etc. These properties overlap and crosscut agent's basic functionality. They should be designed differently from agent's basic behavior. Architecture of multi-agent systems effects the composition of concerns. In addition it also affects the quality of multi-agent systems. Good quality architecture should be able to support separate handling of multiple properties. No such design has been suggested yet for the development of multi-agent systems that can reduce the complexity of system and develop high quality multi-agent systems. Most of the approaches for development of multi-agent systems are attributed to poor architectural design and regardless of modularizing agent properties in the initial stages. This paper shows how an aspect-oriented architecture can be incorporated to previous object-oriented architecture from the preliminary stages to the implementation stage. Separation of agenthood concerns is achieved by 1) prescribing architectural guidelines for aspectizing agent concerns 2) a set of guidelines to describe how aspect-oriented agent architectures can be designed and implemented 3) finally a case study of multi-agent systems has been conducted to test the approach.

The proposed solution is the aspectization of agent architectures at architectural level. Aspect-orientation modularizes agent-specific concerns unlike the previous approaches that use languages, methods and tools. The aspectization is done with a UML extension named aSideML language that provides two views for representing aspects: 1) architectural view and 2) detailed design view. The crosscutting among agent concerns is separated with the help of architectural aspects whereas the modularization of basic concern is achieved with Kernel component. The first stage of the approach is carried in a stepwise fashion in which the configuration and composition of architectural components is understood. In second phase, the architectural component is refined and the corresponding crosscutting interface is also refined to obtain a detailed design. Finally the system is implemented using an aspect-oriented programming language such as AspectJ.

This paper describes that architectural decisions have greater impact in improving the maintainability of multi-agent systems. Aspect-oriented architecture is better than a mediator-based architecture as it supports the functional encapsulation of the agent's basic functionality. The crosscutting interfaces allow the incorporation of both agenthood and

additional properties to the system's basic functionality in a non intrusive way. Therefore, without having any change in the existing methods, simple object architecture can be transformed into agent architecture. Other benefit of using this architecture is its language independency which is helpful for a large number of developers to deploy it.

An Aspect-Oriented Modeling Framework for Designing Multi-Agent Systems [7]

Crosscutting concerns cannot be captured with the conventional modeling approaches because concerns of a specific agent type have their own goals and actions that crosscut goals and actions of other concerns associated with other agent. These concerns encompass the internal and systemic properties of multi-agent systems. The results of crosscutting are scattering and tangling that replicates goals (and actions) in agent-oriented modeling. They are also anti-reuse and anti-evolution factors in the multi-agent software lifecycle.

Similarly, the agenthood properties of multi-agent systems crosscut the multi-agent system modeling elements: agents, goals, beliefs, actions, and plans. When these properties are ignored in the development of multi-agent systems it results in such a system that lacks uniformity with unclear design and implementation decisions. In addition these modeling approaches are proved to be incapable to cope with the crosscutting nature of some multi-agent system concerns. Therefore, new composition rules should be designed to encompass all crosscutting multi-agent system concerns. Hence, a modeling framework is necessary for modular representation and reasoning of crosscutting concerns in a multi-agent system to provide proper support to software developers.

The traditional agent-oriented abstractions and composition mechanisms are revolutionized with a new aspect-oriented meta-modeling framework. This framework, in addition to be an alternative for previous approaches that are limited to object-oriented and component-oriented paradigms, is independent of agent-oriented modeling languages and consists of three models: the Aspect Model, Agent Model and Composition Model. The Agent-Model consists of a set of fundamental agent-oriented design elements such as goals, actions and plans. The Aspect Model is useful for the design of aspect-oriented modeling languages with the help of concepts, relationships, and properties that are organized around three interrelated conceptual models: 1) Component Model, 2) Core and

3) Join point Model. The last model Composition Model is helpful for providing semantics description of the crosscutting composition mechanisms that is it reflects the ways the aspect affect the agents, their beliefs and actions.

With these models, the multi-agent system properties are explicitly modeled the way basic multi-agent system behaviors are modeled. An aspect-oriented notation has been integrated into ANote language but there is no guarantee whether these sets of composition rules are complete. For this purpose more case studies should be conducted to evaluate how much these composition operators can cover.

An Aspect-Based Object-Oriented Model for Multi-Agent Systems [31]

A heterogeneous environment can have multiple types of agents each having different agency properties and collaborative capabilities. Though agents have states and behaviors like those of objects but agent states comprise of beliefs, goals, actions, plans and their behavior with autonomy, adaptation, learning, and mobility. When software agents are introduced into object models they create a number of complexities as these agent properties and capabilities are intrusive and non-orthogonal. There is a need to manipulate these properties and capabilities with the help of an agent model at he beginning of design.

Existing agent models of software architectures focus on one type of agent. In most of the architectures the agents are considered as objects that makes agent design and implementation non understandable, poorly maintained and not reusable. The software agents cannot be designed properly due to the occurrence of many agency properties at the same time. The agency properties and collaborative capabilities of an agent must be associated with its core state and behavior by the help of special techniques and disciplined ways.

So the agent model presented in this paper of object-oriented systems uses aspect-oriented design and programming. As aspect-orientation is good for modularization, it provides separation of crosscutting concerns among different agency properties and collaborative capabilities. Agents are incorporated into the model in a disciplined and non intrinsic fashion. Thus a multi-agent software is obtained whose design and implementation solutions are properly structured for evolution and reuse. The model is applied on a web-based environment that deals with the development of e-commerce

portals to show the results. The model is good for handling each agency aspects separately and helps facilitate the development of a multi-agent software that is easy to understand, maintain and reuse.

Chapter 3

Problem Domain & Proposed Solution

Chapter 3: Problem Domain and Proposed Solution

3.1 Problem Domain

3.1.1 Nature

Decentralized Agent Platform distributes the services and can avoid the centralized MAS bottleneck. A multi-agent system needs to be dynamically scalable which assures flexibility in agent platforms. Fault tolerant architecture must be inherently available in MASs for continuous service provision.

Considering the literature review, we see that all of the papers write about separation of concerns, aspect-oriented architecture, modeling and code generation for crosscutting concerns but none of them write about the decentralized multi-agent system and crosscutting concerns which come across its development.

Our aim is to put an effort to identify crosscutting concerns in decentralized MAS and their implementation in AspectJ. For this purpose we studied SAGE (Scalable, fault tolerant Agent Grooming Environment), which is a decentralized MAS. It is developed in JAVA and is a highly coupled system. The crosscutting concerns in its code are identified. Aspect-Oriented Design (AOD) of above mentioned system is developed to decrease un-necessary communication between modules. Afterwards, metrics are applied that measure the reliability of both of the AMSs developed in JAVA and AspectJ respectively.

3.1.2 Proposed Research

The focus of this research is to analyze that can we achieve a more reliable system while using AOSD for AI based Agent System? It has been recognized that quality of a good software system is that it should obey the principle of loose coupling [10]. Different developers worked on SAGE in parallel while using different coding standards and techniques which made it complex. At this moment it is very difficult to debug and maintain the code of SAGE [11]. Aspect Orientation is applied to autonomous decentralized AI system to observe the behavior of this new methodology.

3.2 Implementation Impacts of Proposed Research

The aim of the proposed research is to explore the behavior of Aspect-Oriented Technology for decentralized Agent-Management System of SAGE.

- This research helps to improve the reliability of decentralized MAS.
- This research is refusal to the idea of researcher or programmers who believe that AOP is good just to modularize the crosscutting concerns like authentication, logging, persistence.
- The crosscutting concerns which we identified in decentralized multi-agent system could never get the attention of researchers previously.
- The issues like code-tangling and code-scattering could be easily handled while using Aspect-Oriented Software Development for an AI based agent system.
- Our research work gives a new dimension to other researchers.

3.3 Functions of Decentralized Agent-Management System

SAGE is following distributed architecture with decentralized management for fault tolerance of multi-agent systems. Its architecture is a blend of merits of client/server and peer-to-peer paradigm. Instead of having centralized location for management the owner-ship rights have been distributed to peer entities which are solely responsible for their roles and actions. These peer entities are part of single Agent Platform and are managed by the Agent Management System. SAGE also introduces a notion of Virtual Agent Cluster (VAC) which plays an important role for introducing fault tolerance within the distributed multi-agent system. Figure 3.1 describes the concept of Virtual Agent Cluster [16]. It proposes a design in which the components of AMS are distributed in such a way that failure of one instance will not cause side effects on its peer instances. AMS communicate with one another through RMI Communication Layer.

This system was ideal for our case study due to several reasons. First, it is the only multi-agent system that has a decentralized agent management platform and Aspect-Oriented programming behavior has not been proved in this context. Second, scalability and reliability always remained a key attribute for multi-agent systems. Finally, its realization involves a number of core and crosscutting concerns; those are of great importance in case of decentralized multi-agent systems.

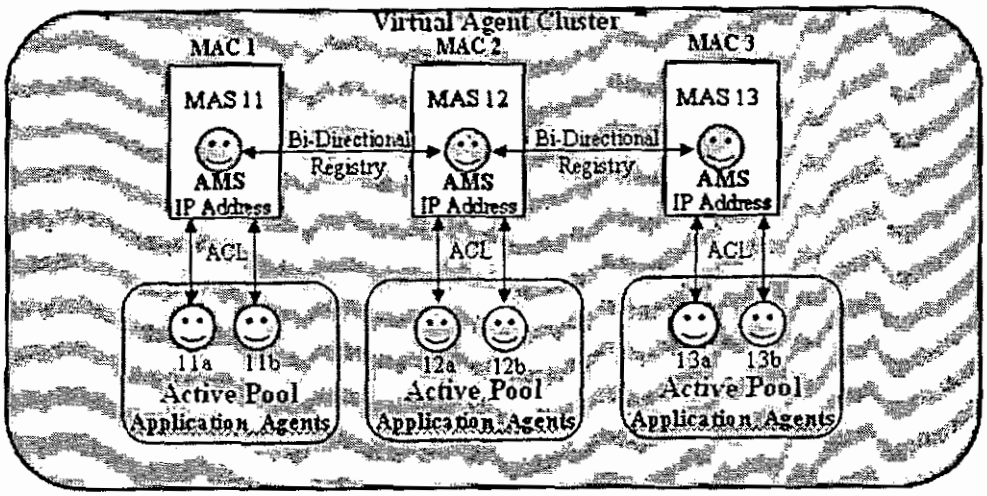

Figure 3.1: Virtual Agent Cluster

3.3.1 Roles & Responsibilities

The AMS is responsible for managing the operation of an Agent Platform, such as the creation of agents, the deletion of agents and overseeing the migration of agents to and from the AP (if agent mobility is supported by the AP) [8]. Since different APs have different capabilities, the AMS can be queried to obtain a description of its AP. A life cycle is associated with each agent on the AP that is maintained by the AMS.

3.3.1.1 Managing Authority

The AMS represents the managing authority of an AP and if the AP spans multiple machines, then the AMS represents the authority across all machines. An AMS can request that an agent performs a specific management function, such as quit (that is, terminate all execution on its AP) and has the authority to forcibly enforce the function if such a request is ignored.

3.3.1.2 Maintaining Index

The AMS maintains an index of all the agents that are currently resident on an AP, which includes the AID of agents. Residency of an agent on the AP implies that the agent has been registered with the AMS. Each agent, in order to comply with the FIPA reference model, must register with the AMS of its HAP (Home Agent Platform).

3.3.1.3 Maintaining Agent Descriptions

Agent descriptions can be later modified at any time and for any reason. Modification is restricted by authorization of the AMS. The life of an agent with an AP terminates with it's deregistration from the AMS. After deregistration, the AID of that agent can be removed by the directory and can be made available to other agents who should request it.

3.3.1.4 Searching Agent Descriptions

Agent description can be searched with the AMS and the AMS further controls access to the directory of AMS agent descriptions; no default policy is specified by FIPA regarding this issue. The AMS is also the custodian of the AP description that can be retrieved by requesting the action get Description.

3.3.1.5 Mandatory Functions of AMS

An AMS must be able to perform the following functions

- register
- deregister
- modify
- search
- get-description

3.3.1.6 Agent Lifecycle

FIPA agents exist physically on an AP and utilise the facilities offered by the AP for realising their functionalities. In this context, an agent, as a physical software process, has a physical life cycle that has to be managed by the AP. This section describes a possible life cycle that can be used to describe the states which it is believed are necessary and the responsibilities of the AMS in these states.

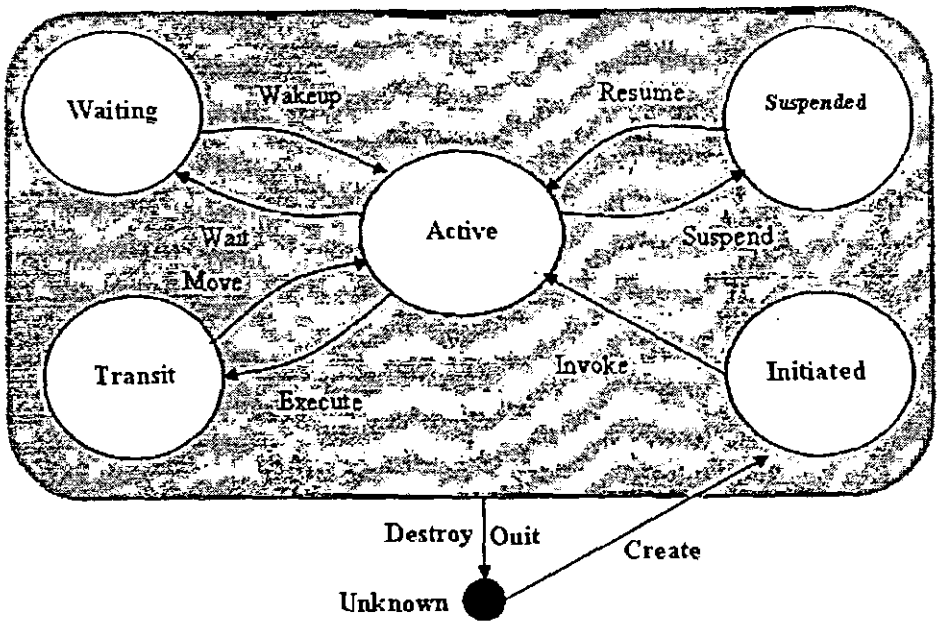

Figure 3.2: Life cycle of a FIPA Agent

In addition to the above methods exchanged between AMS and other agents, AMS can instruct the underlying platform to perform the following operations in order to manage the Agent Life Cycle. Figure 3.2 explains the states of an agent in its Life Cycle [8].

- Suspend Agent
- Terminate Agent
- Create Agent
- Resume Agent
- Invoke Agent
- Execute Agent

3.4 Research Methodology

Our proposed approach is based on the extensive study in the field of multi-agent systems. The basic methodology adopted for the research is:

- Analyzing the decentralized multi-agent system
- Studying SAGE (Scalable, fault tolerant agent grooming environment), its working and design
- Understanding implementation of its decentralized AMS
- Identifying crosscutting concerns in the code of decentralized AMS

- Grouping those crosscutting concerns according to their nature
- Modularizing crosscutting concerns with Aspect-oriented software development techniques
- Implementing aspect-oriented design of decentralized AMS with AspectJ
- Choosing suitable metrics to evaluate reliability of both the versions of decentralized AMSs, one developed with JAVA and other one developed with AspectJ

3.5 Research Work

3.5.1 Object-Oriented Design of AMS

The object-oriented version of decentralized AMS of SAGE was implemented using Java programming language. However, Java is good to deal with inheritance and polymorphism, but it may also be the cause of introducing scattering and tangling in the code. When the AMS of SAGE was reverse-engineered to obtain object-oriented design the same condition existed. We analyzed the system with both class diagrams and interaction diagrams. The result showed that due to high inheritance between classes the system became highly coupled. The methods were incoherently calling each other making the flow of program more complex and not understandable. Moreover, the system has become decentralized to achieve fault tolerance and scalability but it was difficult to organize system components into modules according to their functionality. Therefore, there was a pressing need to implement the system with a technology that could provide linguistics mechanisms for separate expressions of concerns in AMS and weaving these concerns with the system's primary concerns. Figure 3.3 shows the object-oriented design of decentralized agent management system of SAGE which we made by reverse engineering of the system.

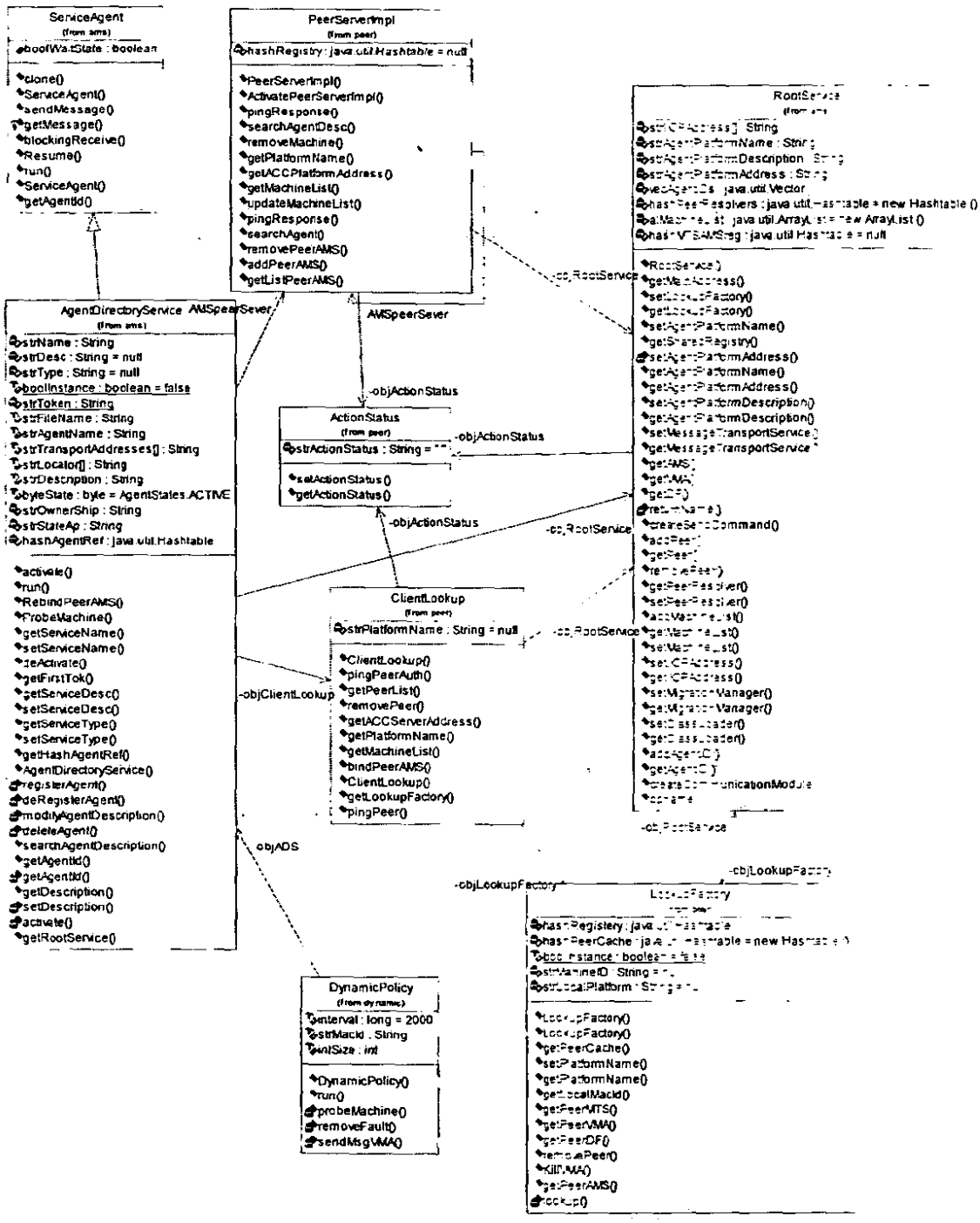

Fig 3.3: Object-Oriented Design of Decentralized AMS

3.5.2 Making an Aspect-Oriented Design

Aspect orientation is the recognisation of the development of software system with respect to many concerns [12]. The development of software system with respect to concerns could be achieved by separating classes into concerns (or modules) according to the agenthood properties they possessed. The Aspect Oriented Design (AOD) [24] was developed then by identifying the crosscutting among these concerns (or modules).

3.5.3 Identification of modules in AMS

Every multi-agent system has a desired set of properties [6] (or agenthood properties) and it is developed in order to achieve those properties. These properties are knowledge, mobility, learning, etc. Therefore, there is a need to go through the AMS in terms of these properties i.e., to see which part (module) of system is related to which agenthood property. It is more likely to analyze properties in a single logical Agent Management System distributed over multiple machines. Therefore. the basic modules or concerns identified according to these properties are:

- Peer Management
- Knowledge Base
- Agent Management

Here is the description of these modules:

Peer Management module:

The Peer Management module is responsible for managing all the operations of peer entities:

1. It manages heartbeats or liveliness of peer machines after a fixed interval of time provided it is done dynamically.
2. It provides interface to the peer entities through RMI (client/server) layer. So any request to server made by the client of peer machines will be acknowledged by the peer management module.
3. It helps peer components or entities to share knowledge with each other. For example, searching and providing information about a certain agent on a peer agent's request.
4. It also helps to manage the knowledge component in case of any sort of modification in peer agent's information.

Knowledge Base module:

Though Knowledge base component manages local agent and remote agent information, it is managed by Peer Management module. Its functions are:

1.2 To keep information about agents i.e., their state, life cycle, etc.

2.2 To perform functions like searching information about a particular agent, deleting, adding or modifying the agent information and providing the information about Agent Platform (AP) description

Agent Management module:

The responsibilities of Agent Management module are similar to those of AMS's responsibilities: register, deregister, managing life cycle and states of agents, etc.

Figure 3.4 represents the functions of these modules.

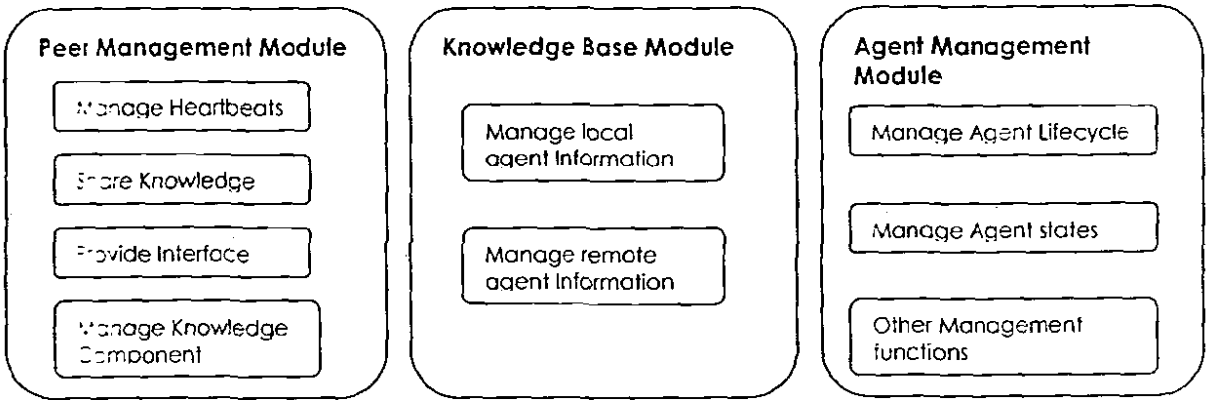

Fig 3.4: Peer management, Knowledge Base and Agent Management modules of AMS

3.5.4 Identification of Crosscutting Behavior

In order to identify the crosscutting behavior, the whole system of AMS is reverse engineered. Its code is analyzed keeping in mind the agenthood properties to identify the modules (Peer management, Knowledge base and Agent management) and then crosscutting is observed in the system. Thus we bring out the classes, members and parts of code in methods/classes according to the behavior of modules. Figure 3.5 represents classes in each module and crosscutting among them.

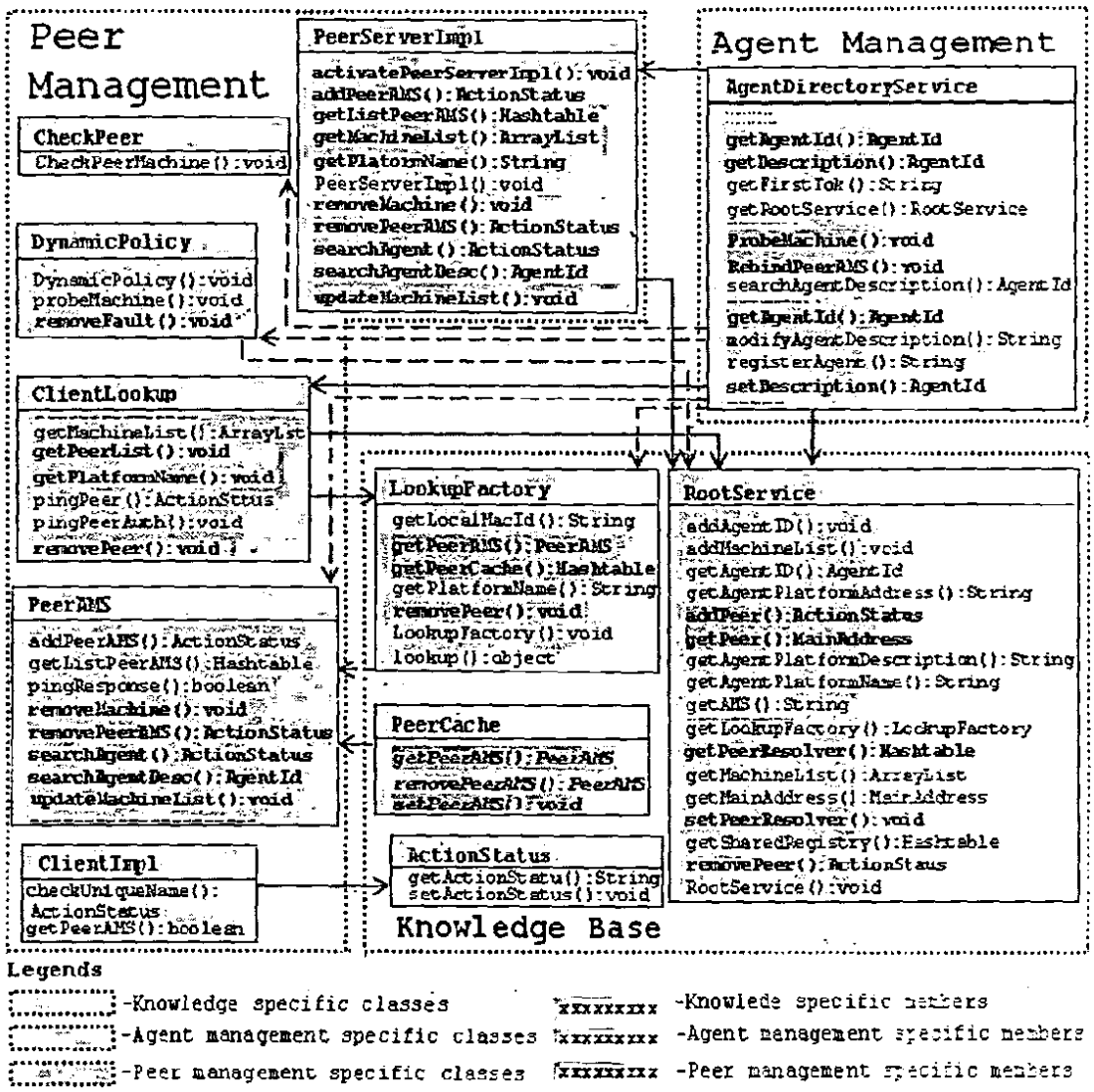

Fig 3.5: Crosscutting Agent Concerns in Peer Management, Knowledge Base and Agent Management Modules

It could be observed that the set of classes belonging to a particular concern (peer management, knowledge base, and agent management) are surrounded by dashed lines while the crosscutting members in a particular class are highlighted with color of concern (to which they belong). The difference between crosscutting members and part of code could be easily noted with some members written in Italic.

From the discussion of above section and through reverse engineering of system, it could be concluded that Knowledge Base is the concern whose functions are crosscutting all the modules of AMS. The functions of Knowledge Base concern are as follows:

1. Updating the knowledge in case of any agent management function
2. Informing other machines about this updated knowledge

Chapter 4

Software Design

Chapter 4: Software Design

4.1 Aspect-Oriented Design of Decentralized AMS

The modules identified in previous section 3.5.2 helped us to identify the crosscutting in the system more appropriately. The main root of crosscutting was Knowledge Base module whose management functions were spread over Peer Management and Agent Management modules. The reason was the shared registry information due to which it was difficult to maintain the distribution of information or updating the information in case of a change in registry, in a peer-to-peer paradigm. Using bidirectional RMI with a combined client/server at both ends does not help in reducing coupling. In addition, consistent registry information at all peer entities brought

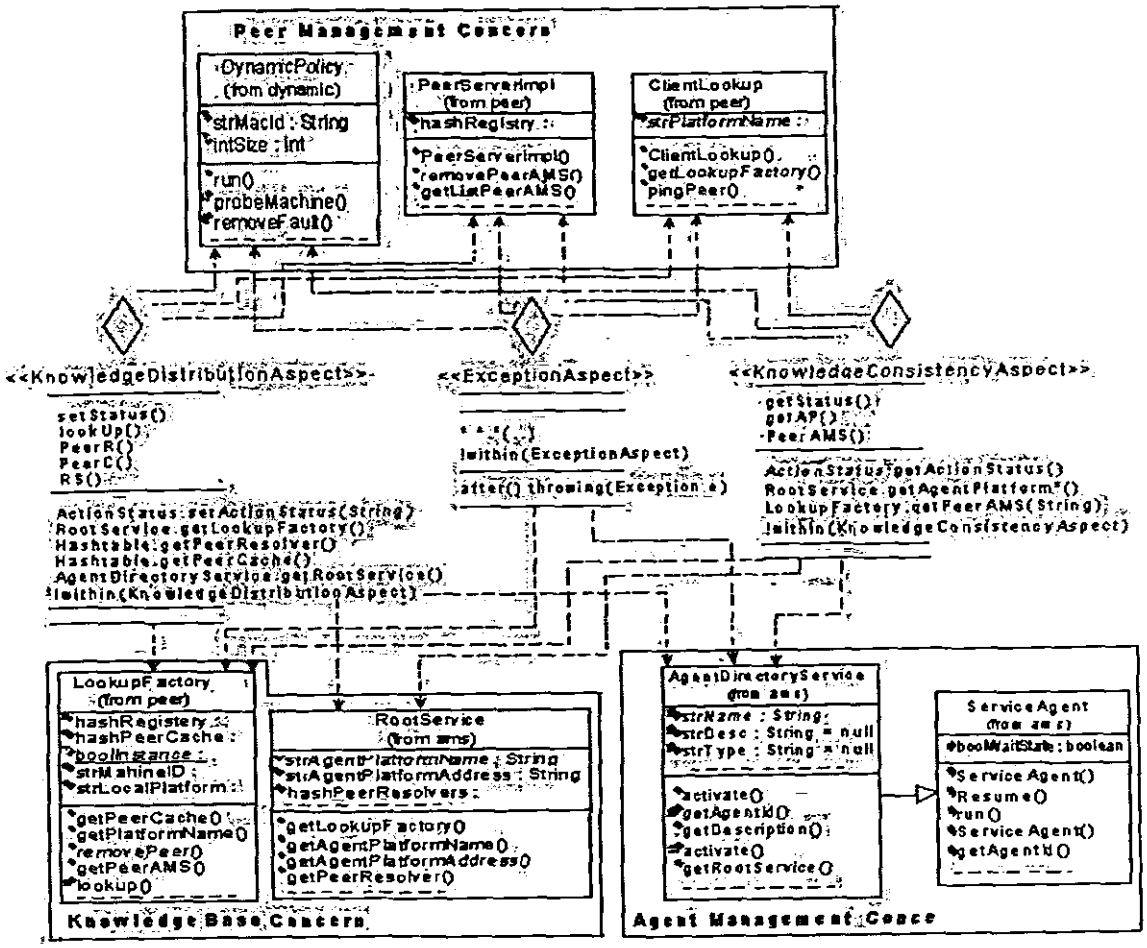

Fig 4.1: Crosscutting Agent Concerns in Peer management, Knowledge base and Agent management modules

the issue of reliability. These issues were resolved by identifying three main aspects that were Knowledge Distribution, Knowledge Consistency and Exception Handling aspects.

Exception handling concern was selected because there were roughly 63 instances of exception handling out of 25 classes in AMS. That is the percentage of Exception Handling was 252% in the system. Therefore, it was necessary to select Exception Handling as a crosscutting concern in order to reduce tangling from the code. The Figure 4.1 shows the AOD of AMS. It shows which class of a module aspect is crosscutting. For example, KnowledgeConsistencyAspect crosscuts all the classes in Peer Management, Knowledge Base and Agent Management modules. Note that classes in each module are selected on the basis of percentage of functionality of a particular module they possessed.

The aspect-oriented version of decentralized agent-management system of SAGE was implemented using AspectJ [4].

4.2 Class Design

The purpose of Peer Management, Knowledge Base and Agent Management functions of AMS is explained and that how they affect the system. Now our focus is to explain the purpose of each class in each module.

4.2.1 Object Oriented Class Design

As we know that OO design of AMS is already developed [16] so the classes in each module are as follows:

- **Dynamic Policy**

  Purpose: An important class of Peer Management module. This class is responsible for the dynamic probing (peer to peer). It checks the status of the other machines on the agent platform and in case of failure it removes the peer machine from the platform.

- **PeerServerImpl**

  Purpose: This is class is also part of Peer Management concern. This is the server class of AMS RMI.

- **ClientLookup**

  Purpose: This is class is also part of Peer Management concern. This is the Client representative class of the AMS RMI. It calls the AMS Server Methods.

- **LookupFactory**

  Purpose: This is class is also part of Knowledge Base concern. Lookup factory is used in booting. LookupFactory is created by the ClientLookup. Lookup Factory is set and can be obtained from *RootService*. It is required by all the system Agents because they need to communicate with other peer System Agents.

- **RootService**

  Purpose: This class is also part of Knowledge Base concern. Its purpose is to provide services to lower level services. In SAGE RootService is responsible for providing services to Agent Platform Services such as AMS, DF, VMA and MTS. Furthermore it also sets service parameters. RootService is initiated at BootStrap time. RootService is set, in which the reference of the shared registry is passed on to the root service instance. It sets the agent platform description. RootService sets the agent platform name, which is obtained by the system and is the name of the system.

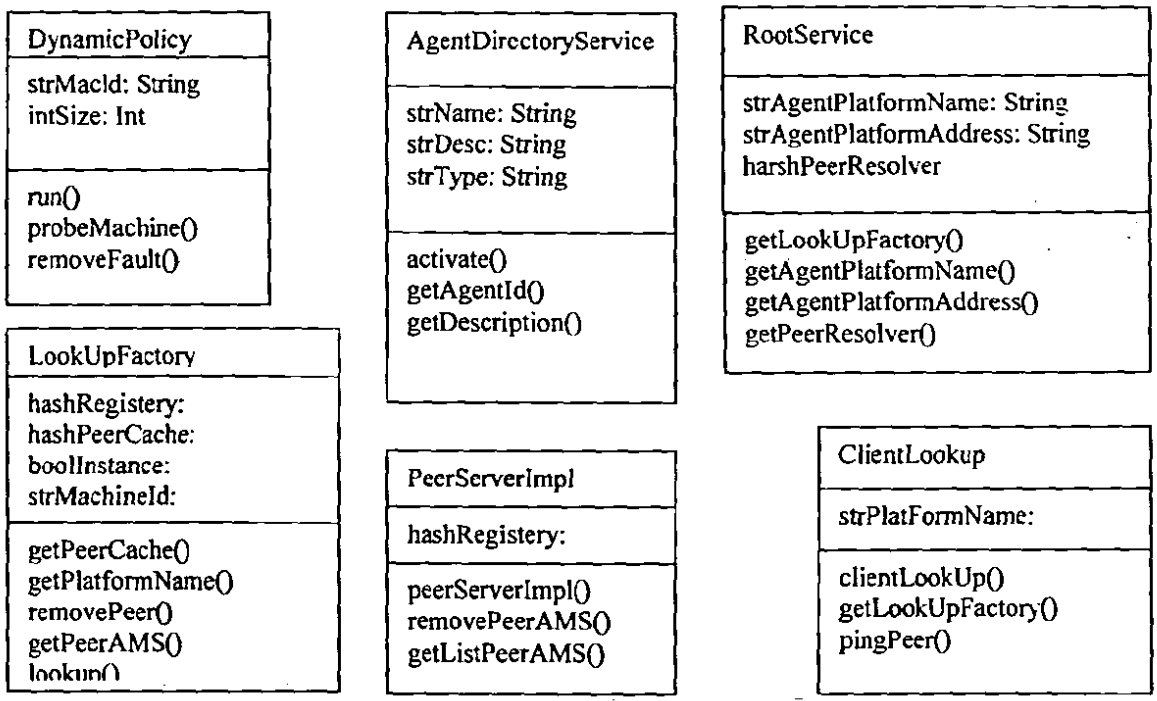

Fig 4.2: Classes with Attributes and Methods

4.2.2 Aspect Oriented Class Design

- **KnowledgeDistributionAspect**

  Purpose: This aspect is associated with the classes of Peer Management concern, Knowledge Base concern, and Agent Management concern. It captures the joinpoints

of RootService AgentDirectoryService and LookupFactory classes from the whole system.

- **KnowledgeConsistencyAspect**

  Purpose: This aspect is associated with the classes of Peer Management concern, Knowledge Base concern, and Agent Management concern. It captures the joinpoints of RootService and LookupFactory classes from the whole system.

- **ExceptionAspect**

  Purpose: This aspect is associated with the classes of Peer Management concern, Knowledge Base concern, and Agent Management concern. It captures joinpoints of exception handling from all the 25 classes of the sytem.

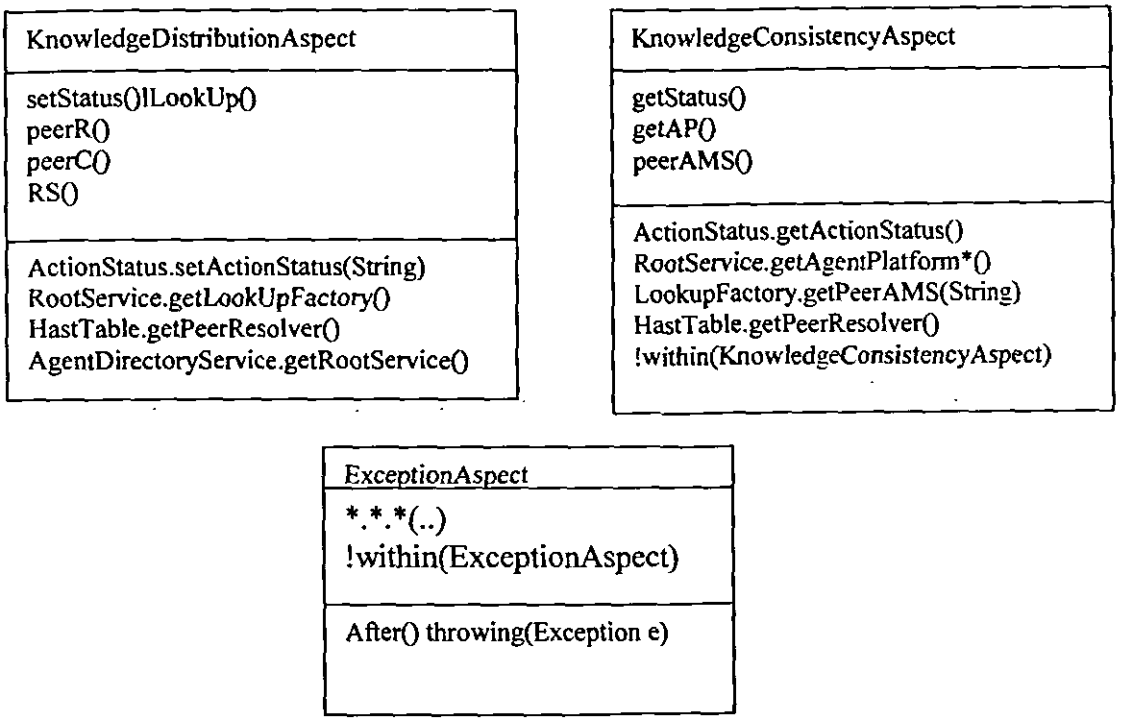

Fig 4.3: Aspects in AMS

4.2.3 UML Design

UML methodology is selected to understand the working of decentralized AMS and to identify the crosscutting between the modules. The following diagrams show the basic design of decentralized AMS.

- **Use-Case Diagram**

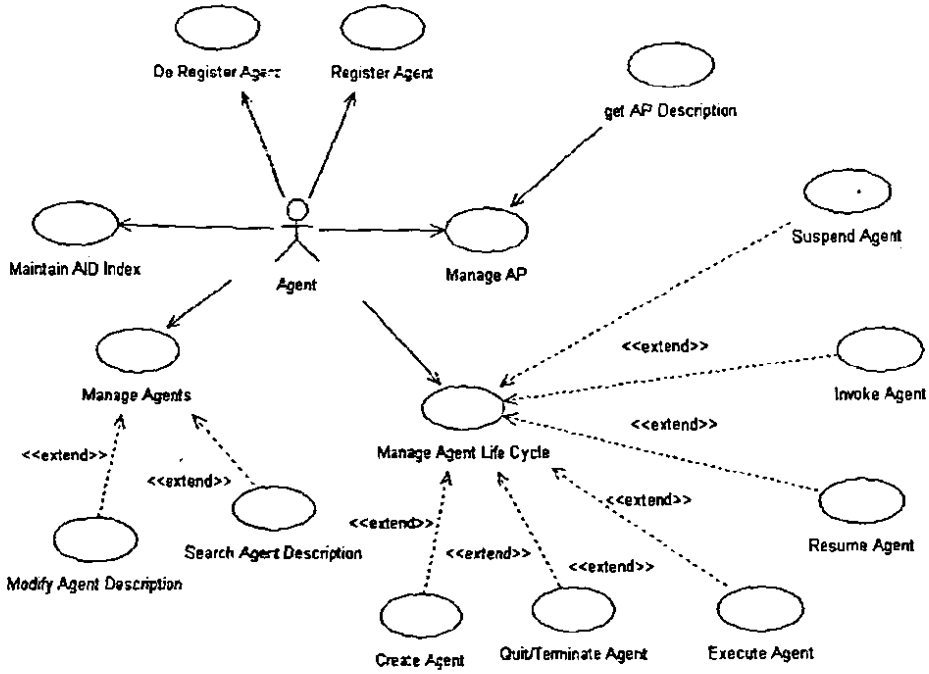

Fig 4.4: AMS Use case Model

- **Sequence Diagram**

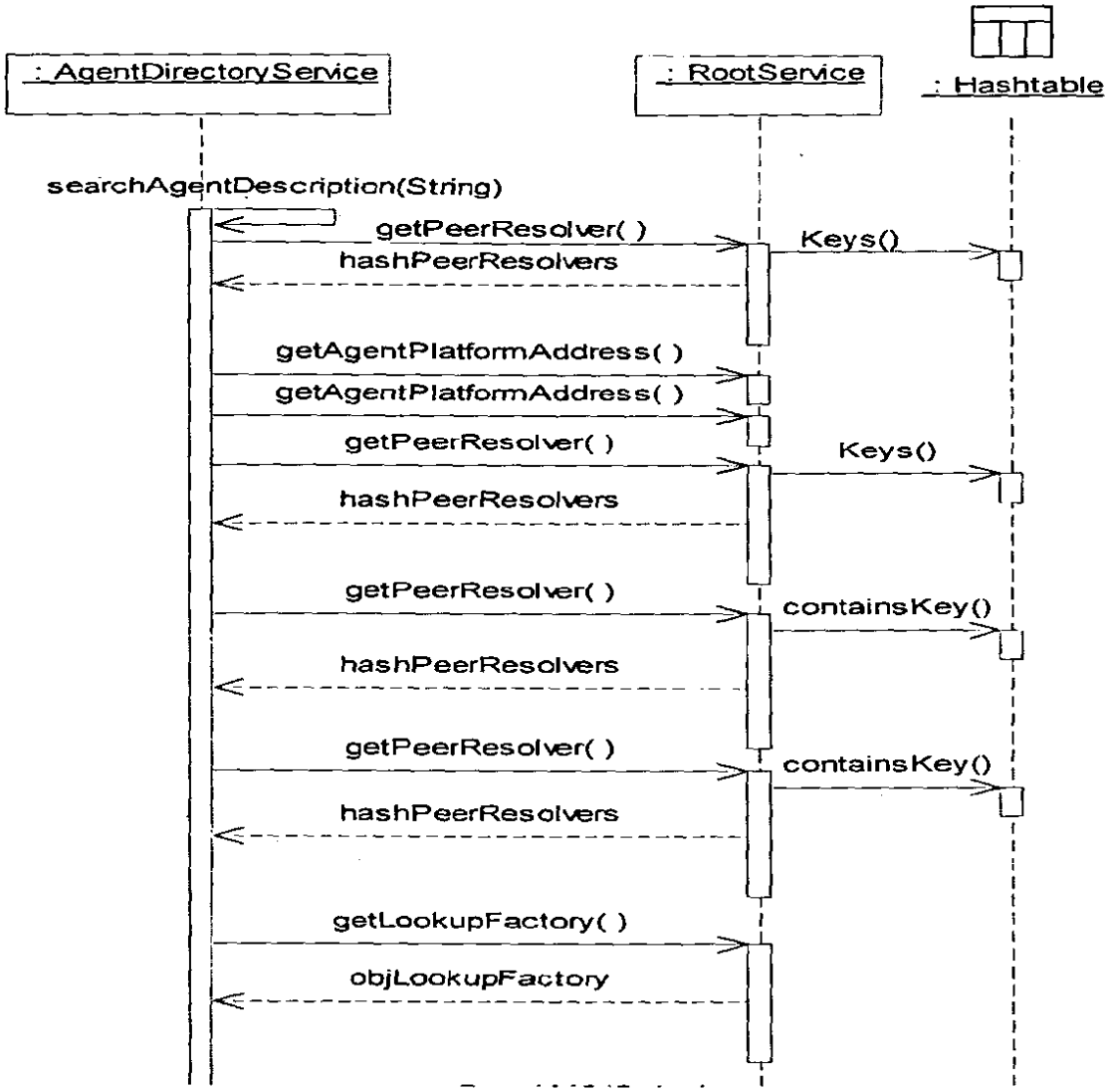

Fig 4.5(a): Interaction Diagram of Search Agent Description

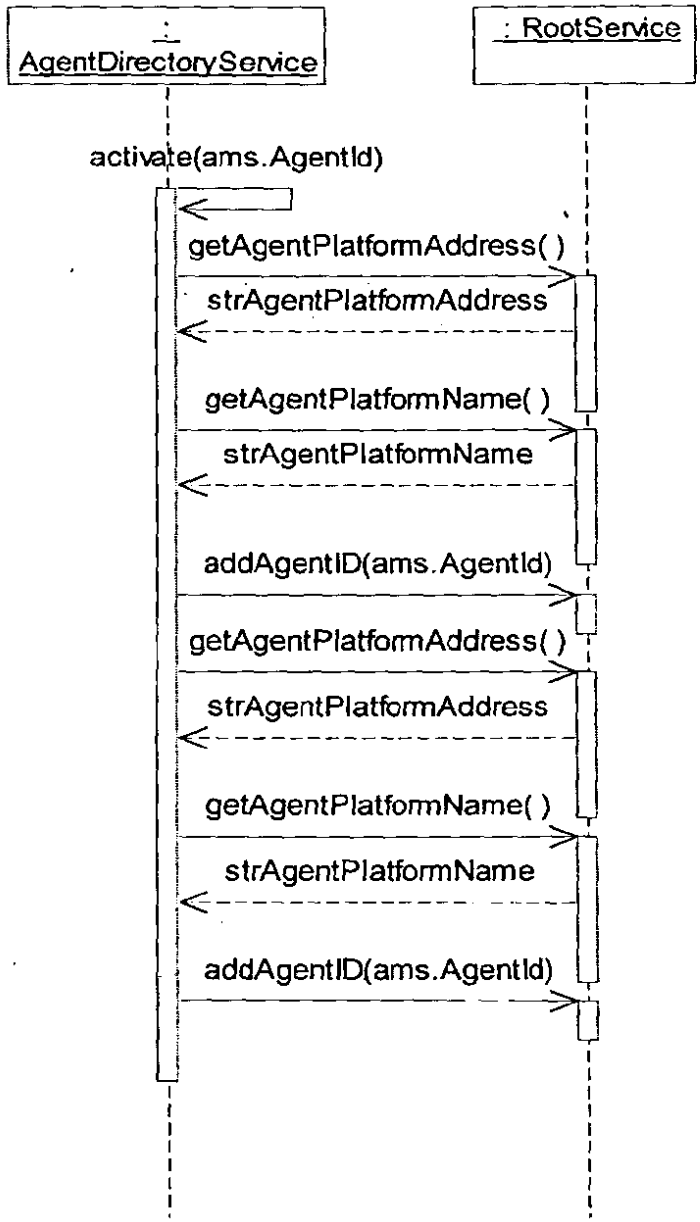

Fig 4.5(b): Interaction Diagram of Activate (Agent State Transition)

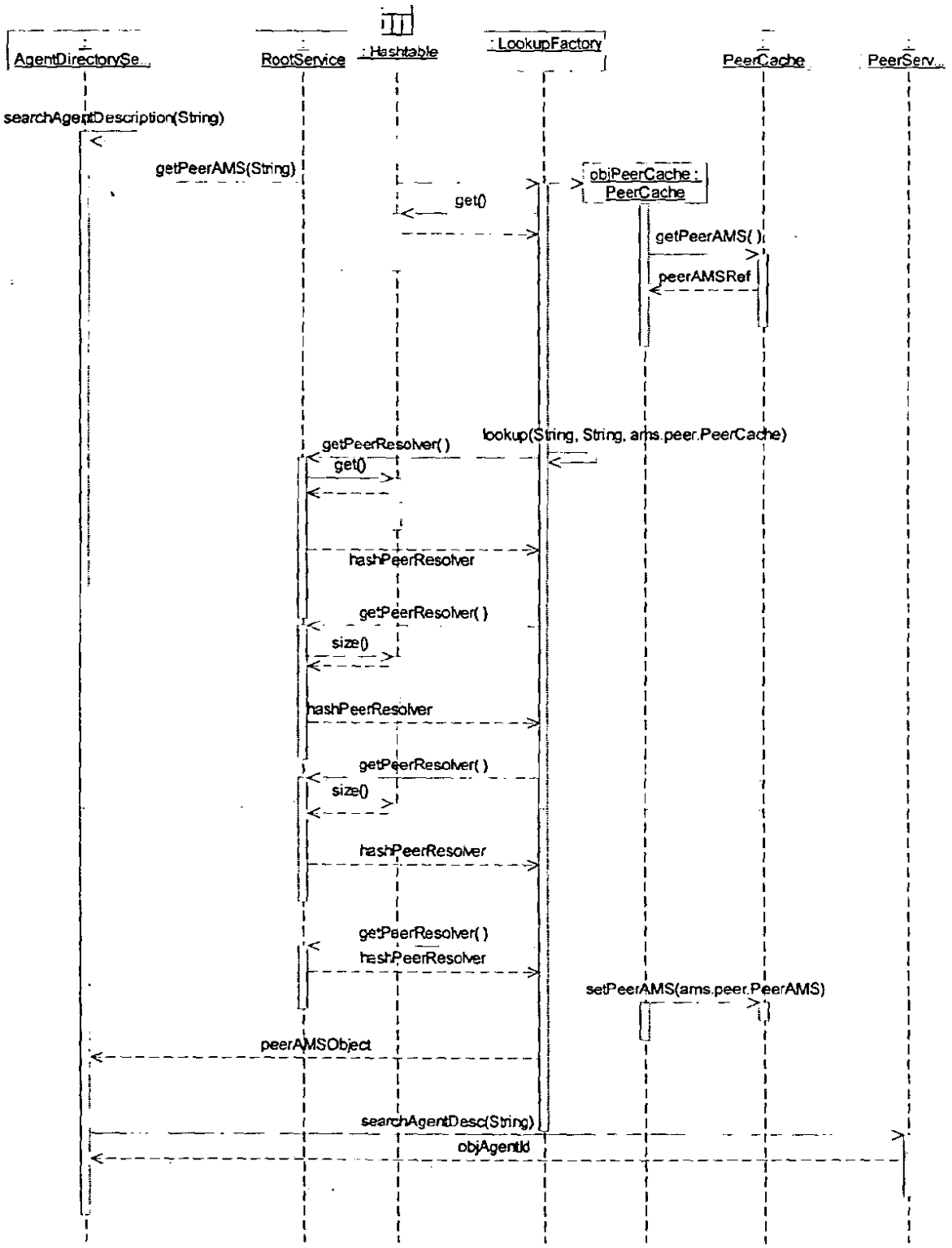

Fig 4.5(c): Interaction Diagram of Search Agent Description

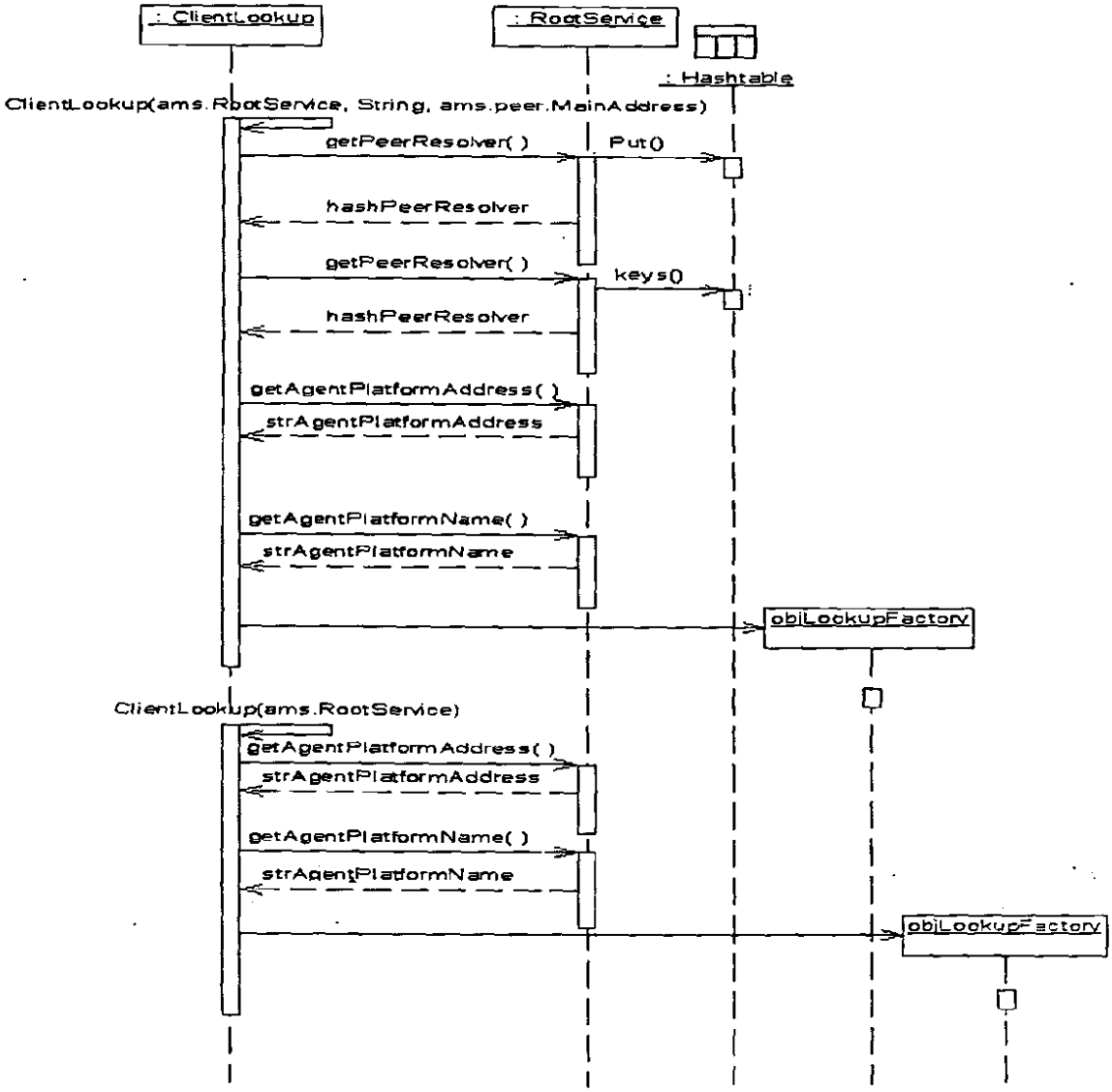

Fig 4.5(d): Interaction Diagram of Client Lookup

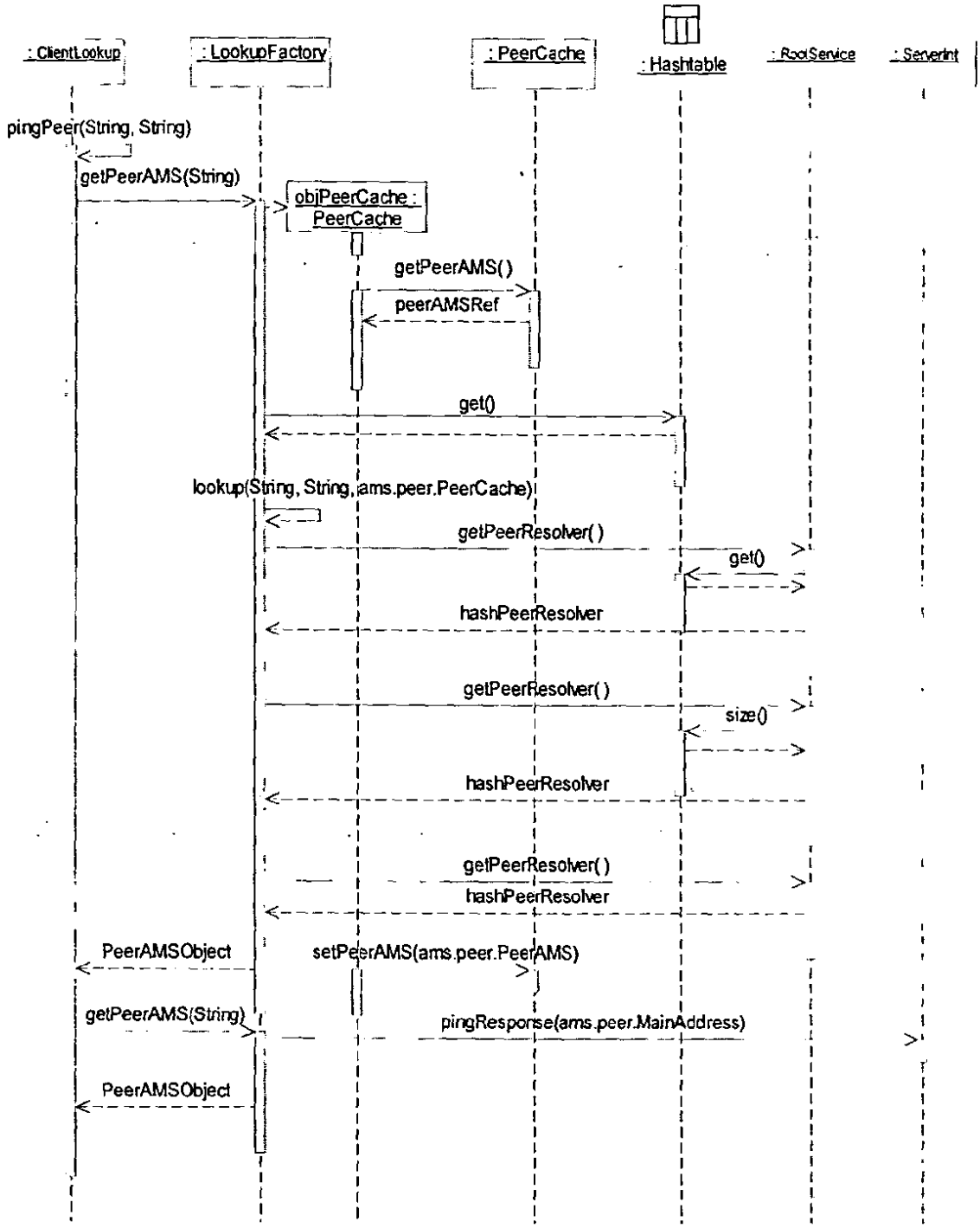

Fig 4.5(e): Interaction Diagram of Ping Peer (Machines)

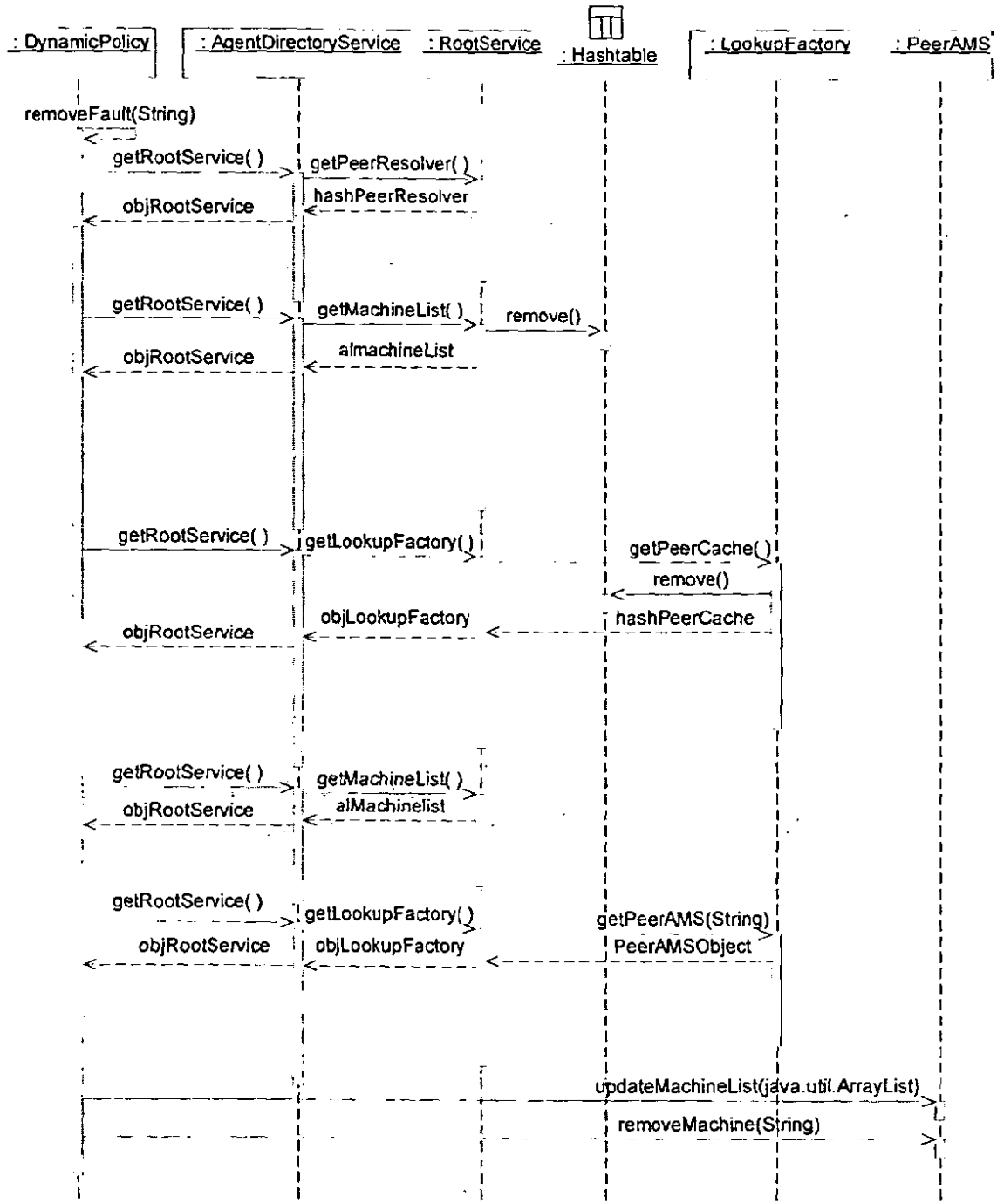

Fig 4.5(f): Interaction Diagram of Remove Fault (Faulty Machine)

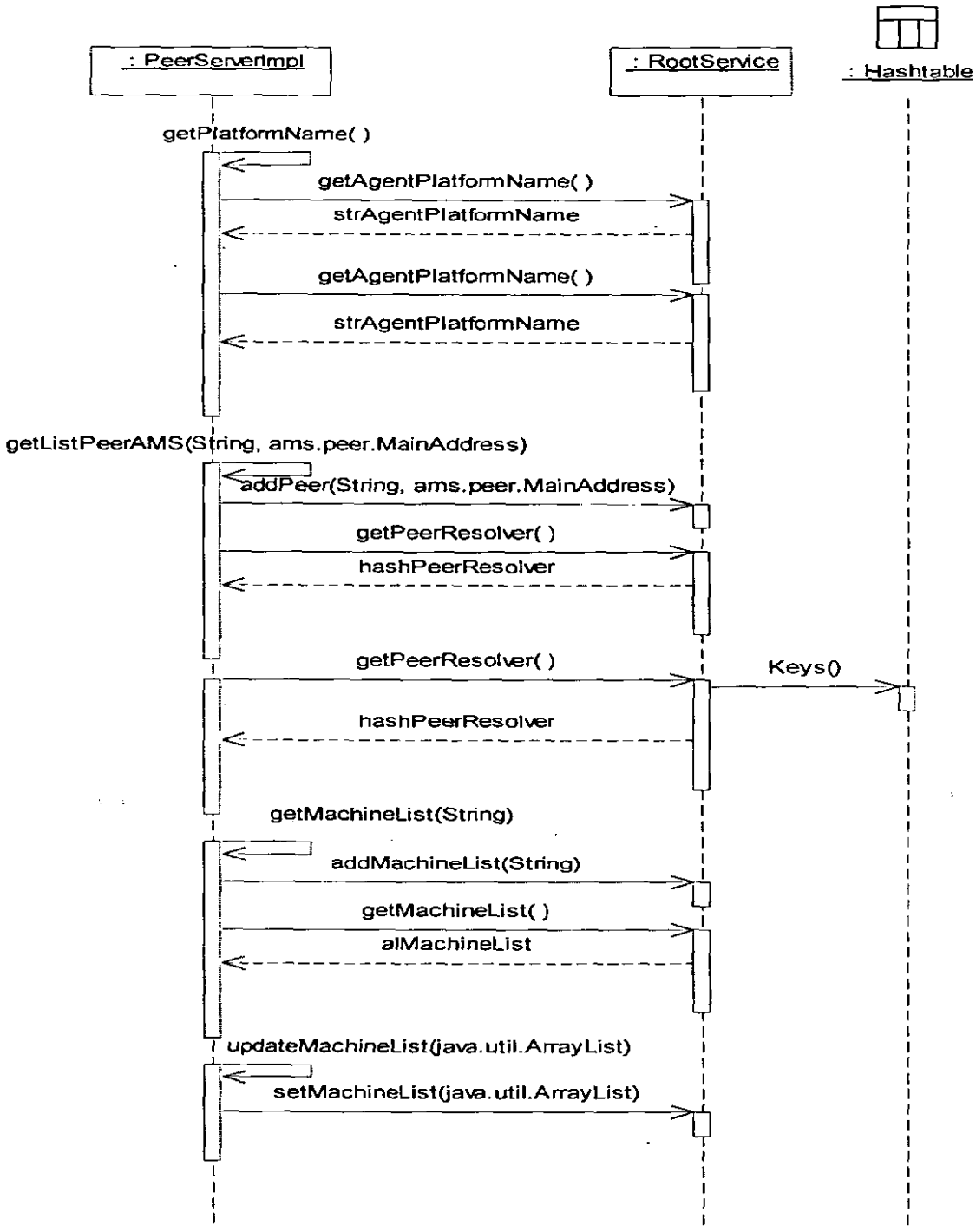

Fig 4.5(g): Interaction Diagram of Updating Machine Information

4.3 Tools to be Used

- Eclipse SDK 3.1.0
- Rational Rose 2002.05.00
- AspectJ Development Tool 1.3.3 for Eclipse 3.1

4.4 Resources Required

- Internet Resources (journals, articles, research papers)
- Books and Magazines
- Printing Resources
- Software CD's
- Computer System

Chapter 5

Software Development & Evaluation

Chapter 5: Software Development and Evaluation

5.1 Software Development

5.1.1 Classes

The following classes are created for the development of software.

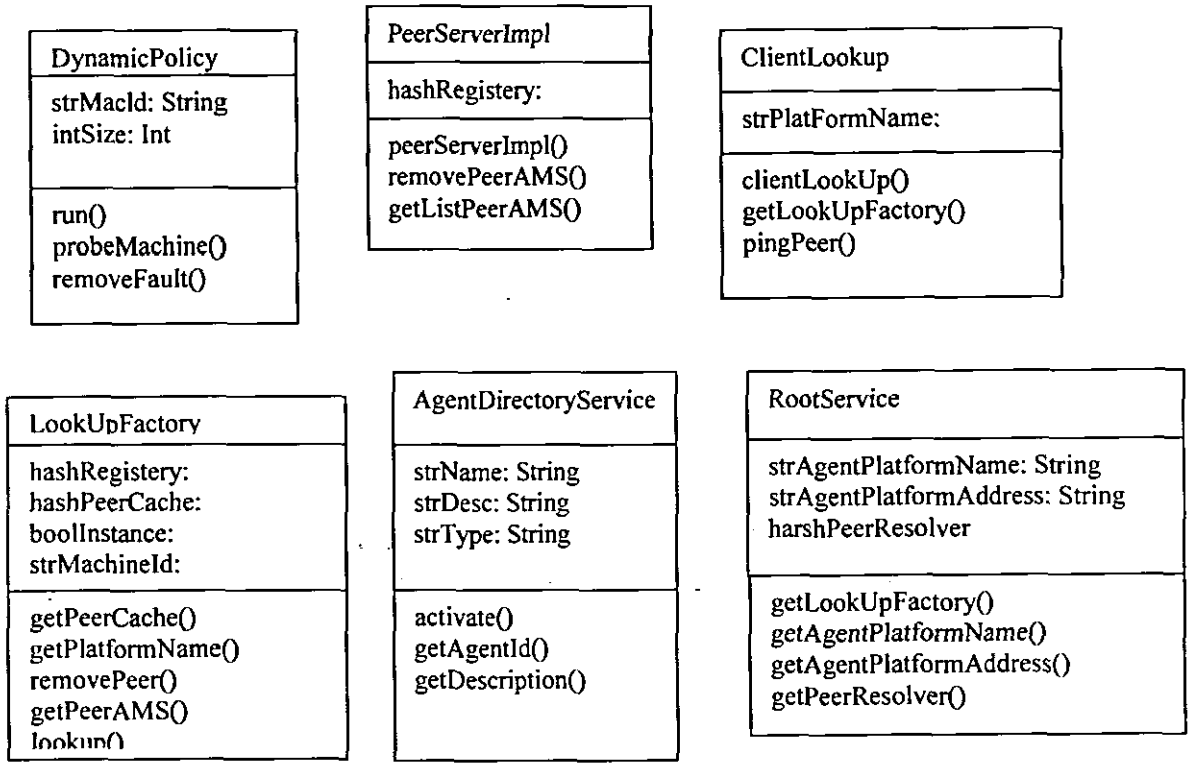

Fig. 5.1: Classes with Attributes and Methods on Development Time

5.1.2 Aspects

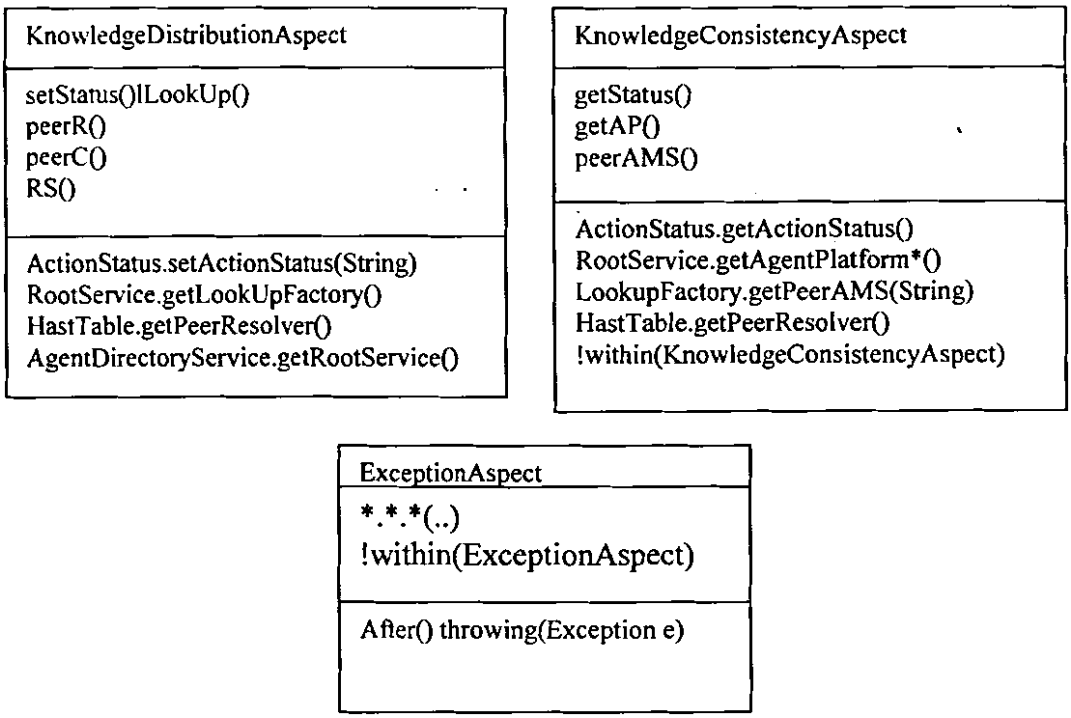

Fig 5.2: Aspects in Decentralized AMS

5.1.3 Software Interfaces

The graphical user interface of AMS visual manager, which allows AMS to perform all of its management related functions, is composed of six main components. These are:

- Agent Tree
- Menu Bar
- Tool Bar
- Popup Menu
- Status Bar
- Menu Bar

Using these components user can interact with the agent and can perform multiple actions which are discussed later in user manual attached for reference. (Refer to Appendix C)

5.1.4 Class Code

The sample code is attached for reference. (Refer to Appendix D)

5.2 The Metrics

Reliability can be measured by measuring the internal attributes like coupling, cohesion and complexity of a system [25] [26] [27]. Chidamber and Kemerer metrics suite [28] are best to capture the above mentioned attributes of software that is why we are using a metric suite from [29], which is based on the refinement of CK metrics for aspect-oriented systems and is reusing LOC metrics. These metrics capture the degree to which a single system concern maps to design components (classes and aspects) and operations (methods and advice) [5]. The chosen metrics were applied on class diagrams of both the versions and where it was needed, behavioral diagrams were also consulted. Table 5.1 briefly defines each metric and associates it with the relevant software attribute. We grouped the metrics to measure a certain attribute, according to our own requirements. The reason behind measuring above mentioned internal attributes is, to see which version has more error-prone and afterwards we applied Mean Time to Failure on both versions to see how these attributes affect system's reliability.

Attribute	Metrics	Definition
Coupling	Coupling Between Components (CBC)	Counts the number of other classes and aspects to which a class or an aspect is coupled.
	Depth of Inheritance Tree (DIT)	Counts how far down in the inheritance hierarchy a class or aspect is declared.
Cohesion	Lack of Cohesion in Operations (LCOO)	Measures the lack of cohesion of a class or an aspect in terms of the amount of method and advice pairs that do not assess the same instance variable.
Complexity	Weighted Operations per Component (WOC)	Counts the number of methods and advice of each class or aspects or the number of its parameters.
	Lines of Code (LOC)	Counts the lines of code.
	Number of Attributes (NOA)	*Counts the number of attributes of each class or aspect.*
MTTF	Mean Time To Failure	It measures the average time between observed system failures.

Table 5.1: The Metrics Suite

5.3 Evaluation

Table 5.2 and 5.3 present the computed metric values for both AO and OO versions. We compared both AO and OO systems on the basis of each metric value as follows:

Coupling

- Coupling Between Components (CBC)

From the metric values, it can be noticed that in AOD the coupling is increasing but if we observe the AOD in Figure 4.1, we can understand that the coupling between core classes is decreasing and the coupling between core classes and aspects is increasing.

- Depth of Inheritance Tree (DIT)

Only one class with the name of ServiceAgent has a subclass of AgentDirectoryService which does not pay a remarkable effect on a system.

Cohesion

- Lack of Cohesion in Operations (LCOO)

Values for the LCOO decreased in aspect-oriented design, which means the AO version of the system is more cohesive as compared to OO version. System with more cohesion is more reliable and efficient.

Complexity

- Weighted Operations per Component (WOC)

As it can be seen from the results of metrics, that in AO version, the number of operations per class/aspect is reduced as compared to OO version. Aspect-oriented software development helps in decreasing the crosscutting between the classes and reduces the number of tangled methods in the class. Therefore it decreases the overall complexity of a system.

- Lines of Code (LOC)

LOC is 1685 in the OO implementation and 1486 in AO implementation. This shows that OO system is more complex in terms of LOC.

- Number of Attributes (NOA)

It can be noted that object-oriented version is more complex in terms of NOA, while NOA is reduced in aspect-oriented version.

METRICS		COUPLING		COHESION	COMPLEXITY		
		CBC	DIT	LCOO	WOC	LOC	NOA
Class Name	DynamicPolicy	1	0	6	5	145	7
	AgentDirectoryService	9	0	12	10	376	14
	PeerServerImp	3	0	11	14	227	1
	RoolService	8	0	22	35	347	8
	ClientLookup	3	0	5	11	159	1
	LookupFactory	2	0	15	13	299	5
	ServiceAgent	0	1	4	9	132	1

Table 5.2: Metrics Obtained For OO Design

S. No.	Class Name	COUPLING		COHESION	COMPLEXITY		
		CBC	DIT	LCOO	WOC	LOC	NOA
1	[illegible]Policy	3	0	2	3	133	5
2	AgentDirectoryService	3	0	9	6	302	11
3	[illegible]	3	0	5	9	192	1
4	[illegible]	2	0	13	22	258	5
5	[illegible]	3	0	2	7	133	1
6	[illegible]	3	0	8	11	231	5
7	[illegible]	0	1	9	13	130	1
8	KnowledgeDistributionAspect	6	0	5	6	48	5
9	[illegible]	5	0	3	1	22	3
10	KnowledgeConsistancyAspect	6	0	7	4	37	3

Table 5.3: Metrics Obtained For AO Design

5.3.1 Reliability

Reliability is the probability of failure-free operation over a specified time in a given environment for a specific purpose. It is a complex concept which should always be considered at the system rather than the individual component level. Software reliability is the probability that how likely a software component will produce an incorrect output. As the number of dependent components increase the overall probability of system failure increases.

Reliability is a quality attribute and can be divided into two categories those are:

- Fault tolerance
- Maturity

Fault tolerance can be achieved by two ways, one is defensive programming and other one is by fault tolerant architecture. In aspect-oriented version of decentralized AMS of SAGE, fault tolerance is achieved by defensive programming and that's why exception handling was taken as primary cross-cutting concerns while Maturity which is the second category of reliability, is measured by using metrics Mean Time to Failure (MTTF).

Reliability can be quantitatively measured by using probability of failure on demand, rate of failure occurrence and mean time to failure. We used MTTF to evaluate reliability of both the systems. MTTF is the average time between observed system failures. An MTTF of 500 means that one failure can be expected every 500 times units. This metric should be used for the systems where there are long transactions, i.e. where

people use system for a long time [30]. That is why we chose this metric to evaluate our system. The next section explains the measurement of MTTF for our system.

5.3.2 Mean Time to Failure

MTTF is calculated by observing how much longer resources SAGE occupies in a platform at the time of its operation. We took Memory Usage and CPU Usage as the main resource. So the readings for both OO and AO systems were taken and evaluated.

Readings for Object Oriented Version of SAGE

Memory Usage at the point of System Initialization: 35028 KB.

Agents started at initial stage VMA, AMS and DF.

Additional Agents created DF GUI and Test Agents (23 in total).

Change in Memory Usage after creation of 27 agents: 39964 KB.

So the agents were as follows:

Memory Usage at the start of GUI of 24th Test agent: 41448 KB.

At the point when all agents were selected to send messages 42044K. Table 5.4 shows the change in memory usage every time an action is being taken on SAGE

Time	No of Agents Created	Total Agents	Memory Usage (KB)
Start of SAGE GUI	3	3	35028
Additional Agents Created: DF GUI Test Agents	1 23	27	39964
Start of GUI of 24th Test Agent	1	28	41448

Table 5.4: Memory Usage of OO system at the System Initialization point

Sending Messages:

First Message

Time Started 7:40 pm

Total Agents: 28 (1 itself to count how much messages were sent)

Total messages sent one time: 27

Memory Usage: 42316 K

CPU usage 11% from 4-5% normal

Then a number of times messages were sent until the system became halted and could not accept any more messages. The reading is as follows:

Halting Time 7:42 pm

No of times messages sent: 356

Total messages sent : 9612

Memory Usage : 96832 K

CPU Usage : 100->72%

It was observed that system was not fully occupied by its resources even the CPU usage reached 100%, the system would release itself decreasing the CPU usage to 72%. Therefore, at this point some more messages were sent until CPU usage became constant to 100% and system reached deadlocked state. These readings are shown in Table 5.5.

No of Time Messages Sent	No Of Agents	Total Messages	Memory Usage (KB)	Time	CPU Usage
1	28	27	42316	7:40 pm	11%
356	28	9612	96832	7:42 pm	100%
358	28	9666	96840		100%
362	28	9774	96892		100%

Table 5.5: Memory Usage and CPU Usage of OO system on Sending Messages

Readings for Aspect Oriented Version of SAGE

For AO same system was restarted to take the fresh readings. Table 5.6 shows the readings. Memory Usage at the start of SAGE GUI: 27952-964 KB at the time when 3 agents are created by default. The procedure was same for AO system as well.

Time	No Of Agents Created	Total Agents	Memory Usage (KB)
Start of SAGE GUI	3	3	27964
Additional Agents Created: DF GUI Test Agents	 1 23	27	30672
Start of GUI of 24th Test Agent	1	28	39084

Table 5.6: Memory Usage of AO system at the System Initialization point

Sending Messages:

First Message

Time Started 8:06 pm

Total Agents: 28 (1 itself to count how much messages were sent)

Total messages sent one time: 27

Memory Usage: 39324 K

CPU usage 4-5%

Then a number of times messages were sent until the system reached its first deadlock and could not accept any more messages. The readings are shown in Table 5.7. The time at which the system halted was 8:08 pm and 362 messages were sent at that time consuming memory to 86880KB and increasing CPU usage from 4-5% (normal) to 100%. After a short while the system released its resources decreasing CPU usage to 72%. So some more messages were sent to see the working of system. The CPU usage decreased to 5% that would again rise up to 100% as more messages were sent. But see in Table 5.7 that memory usage almost remained constant though it would rise a little bit but it would again become constant. The performance of system decreased slowly each time a message was sent until it finally reached deadlock state. The readings after the first deadlock are also shown in Table 5.7.

The readings for both OO and AO versions are shown in Table 5.8. The first difference is observed from the Memory usage at the start of GUI of SAGE that is great difference. In addition the AO system even after its first deadlock state was able to send more messages than it OO counterpart. Its performance was better for AO than OO system. The OO system failed to send more messages after it had sent 356 messages while AO system had sent 362 messages at that time and AO system was still able to send 25 more messages while OO system could only send 6 more.

No of Times Messages Sent	No Of Agents	Total Messages	Memory Usage (KB)	Time	CPU Usage
1	28	27	39324	8:06 pm	4-5%
362	28	9774	86880	8:08 pm	100%
368	28	9936	88320		72%
376	28	10152	90240		23%
380	28	10260	98264		5%
381	28	10287	98264		100%
382	28	10314	98300		100%
384	28	10368	98265		21%
385	28	10395	98240		100%
386	28	10422	98272		100%
387	28	10499	98272		100%

Table 5.7: Memory Usage and CPU Usage of AO system on sending messages

		OO System	AO System
Memory Usage	Start of GUI	35028KB	27964KB
	After creation of 27th Test Agent	41448KB	39084KB
No of Times Messages Sent		362	387
No of Messages Sent		9774	10449

Table 5.8: Comparison of Memory Usage for OO and AO system of SAGE on sending messages

Chapter 6: Conclusion

6.1 Research Results and Conclusion

In this research work, we have compared aspect-oriented and object-oriented versions of the same application, in order to explore to what extent each implementation provides a reliable system. A group of researchers have worked on aspect-oriented architecture, modeling and engineering of MASs, but there is no empirical evidence whether AOP helps in improving the reliability of a Decentralized Multi-Agent System, thereby hindering the adoption of AOP for such a system. This was the reason of choosing Decentralized Multi-Agent System for this experiment. The research work is based on a case-study in which we have compared the reliability of aspect-oriented and object-oriented design of Decentralized Agent Management System (AMS) of SAGE.

SAGE (Scalable and fault tolerant agent grooming environment) is FIPA compliant Decentralized MAS. It consists of modules those are large scaled and complex. Our focus was to check the impacts of Aspect-Orientation on a non-trivial system, so we took Decentralized AMS of SAGE. The system has got sub-modules as well. AMS is a large module which was modified for fault-tolerance and scalability. If software undergoes upgradation and changes, it suffers from high degree of failure rates and complexity. Therefore this becomes necessary to make software reliable after upgradation. As AMS was also upgraded for decentralization, it was more likely to be checked for reliability. Fault-tolerance and scalability have incurred increase in failure rates that lead to the maintenance of the system for reliability. Reliability can be achieved if we re-design and re-implement this module of the system by using better engineering approach. Therefore, we used AOP to see its impacts on the reliability of Decentralized AMS, as it was considered better approach than OOP for the development of other systems.

Reliability remained the main concern on which we focused in this research, which is a by-product of quality that could be measured. It can be defined as 'extent to which a program can be expected to perform its intended functions with required precision'. There are a number of metrics those could be used to measure the reliability of a system, which are, Probability of Failure on Demand (POFD), Rate of Failure Occurrence (ROFO) and Mean Time to Failure (MTTF). We used MTTF to measure the

reliability of both the versions of the system because MTTF is best to measure the maturity of a system and it is successfully being used for systems those run for longer time. We evaluated the systems for reliability and observed how coupling and cohesion affected reliability of OO and AO versions of the system.

Aspect-Orientation basically talks about separation of concerns in software construction. It states that a given problem involves different types of concerns, which should be identified and separated in order to manage the complexity of the system. It also provides loose coupling between modules. Here we have tried to relate the reliability of aspect-oriented systems with the internal attributes like coupling, cohesion and complexity. We have used refinement of CK metrics for aspect-oriented system and measured the above mentioned internal attributes; those are basic error-prone and afterwards applied MTTF on both the versions of the system. We observed that aspect-oriented system that showed good results for CK metrics also showed good results for MTTF. By measuring MTTF we observed that Aspect-Oriented systems tries to tolerate the failure as much as it could and keeps Memory Usage constant even when the CPU Usage was very high. Hence it is proved that AOP improves the reliability of Decentralized MAS and therefore software engineers should not be hesitant in using this development technique for Decentralized MASs. This experiment resulted into good impacts of Aspect-Orientation on the reliability of Decentralized MAS.

Appendix A

References

A. 1 References

[1] Garcia, A., Silva, V., Chavez, C., Lucena, C. Engineering Multi-Agent Systems with Aspects and Patterns. Journal of the Brazilian Computer Society, July 2002.

[2] Araujo, J., A. Moreira, I. Brito and A. Rashid. Aspect-Oriented Requirements with UML. Workshop on Aspect-Oriented Modeling with UML, 2002.

[3] Iris Groher and Thomas Baumgarth. Aspect-Orientation from Design to Code. In Proceeding: Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design; AOSD, March 2004.

[4] Ramnivas Laddad. AspectJ in Action. Pages (7-11), Oreilly & Associates Inc., 2003.

[5] U. Kulesza, et al. Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study. In Proceedings of the 9th International Conference on Software Reuse (ICSM'06), Philadelphia, USA, September 2006.

[6] Alessandro Garcia, et al. Aspectizing Multi-Agent Systems: From Architecture to Implementation. PUC-Rio, Computer Science Department, LES, SoC+Agents Group, Rio de Janeiro, RJ, Brazil, 2004.

[7] Garcia, A., Chavez, C., Choren, R. An Aspect-Oriented Modeling Framework for Designing Multi-Agent Systems. 7th Workshop on Agent-Oriented Software Engineering, AAMAS'06, Hakodate Japan, May 2006.

[8] Foundation for Intelligent Physical Agents. http://www.fipa.org

[9] Abdul Ghafoor, et al., SAGE: Next Generation Multi-Agent System, In Proceedings of the 2004 International Conference on Parallel and Distributed Processing Techniques and Applications (USA), June 2004, pp. 139-145.

[10] J. Zhao. Measuring Coupling in Aspect-Oriented Systems. Technical Report, SE-142-6, Information Processing society of Japan (IPSJ), June 2003.

[11] A. Ghafoor, A. Shibli and H. Farooq Ahmad, SAGE, Open Source Fault Tolerant Architecture: Enhancement, Refactoring and Debugging, 21st Assurance System Symposium, Hiroshima City University, Hiroshima, Japan 2007.

[12] Kiczales, G., Hilsdale, E., Hugunin et al., An Overview of AspectJ, In Proceedings of ECOOP 2001, Lecture Notes in Computer Science, Vol. 2072, Springer (2001) 327-353.

[13] R. E. Filman, et al (Eds.) Aspect-Oriented Software Development, Addison-Wesley, 2005.

[14] Katia P. Sycara, Multi-Agent Systems, 1998.

[15] N. Ubayashi, T. Tamai. Separation of Concerns in Mobile Agent Applications. In Proceeding of the 3rd Conference Reflection 2001, LNCS 2192, Kyoto, September 2001, pp. 89-109.

[16] Salman Shahid's "Agent Management System (AMS) for FIPA Compliant Multi-Agent System", Distributed Computing Group NUST Institute of Information Technology, Rawalpindi, 2000.

[17] Garcia, A. From Objects to Agents: An Aspect-Oriented Approach. PhD Thesis, Computer Science Department, PUC-Rio, Brazil, April 2004.

[18] Garcia, A. Separation of Concerns in Multi-Agent Systems: An Empirical Study. In: C. Lucena et al (Eds). Software Engineering for Multi-Agent Systems II. Springer-Verlag, LNCS 2940, February 2004.

[19] M. D'Hondt, K. Gybels, V. Jonckers. Seamless Integration of Rule-Based Knowledge and Object-Oriented Functionality with Linguistic Symbiosis. Proceedings of the 19th Annual ACM Symposium on Applied Computing (SAC 2004), Nicosia, Cyprus, March 2004.

[20] Z. Guessoum, J. Briot. From Active Objects to Autonomous Agents. IEEE Concurrency, Special Series on Actors and Agents, 1999, pp. 68-76.

[21] A. Amandi, A. Price. Building Object-Agents from a Software Meta-Architecture. In: Advances in Artificial Intelligence, LNAI, vol. 1515, Springer-Verlag, 1998.

[22] Kendall, E. Role Model Designs and Implementations with Aspect-oriented Programming. OOPSLA 1999, pp. 353-369.

[23] Carlos Lucena, Paulo Alencar, Alessandro Garcia, A Generative Approach for Multi-Agent System Development (2004).

[24] C. Von Flach and Carlos J.P., Design Level Support for Aspect-Oriented Development, Workshop on Advanced Separation of Concerns in Object-Oriented Systems (ASOC) at OOPSLA' 2001, Tampa Bay, Florida, USA, October 14, 2001.

[25] Dr. Linda Rosenberg, et al, Software Metrics and Reliability, 9th International Symposium on Software Reliability Engineering Germany, Nov 1998.

[26] Jubair J. Al-ja'afer, Khair Eddin M. Sabri, Chidamber-Kemerer(CK) And Lorenze-Kidd(LK) Metrics To Assess Java Programs, In Proceeding of International Workshop on Software Systems (IWSS), 2004.

[27] Jacqueline A. Mc Quillan and James F. Power, On the Application of Softwares Metrics to UML Models.

[28] Chidamber, S. and Kemerer, C., A Metric Suite for Object Oriented Design, IEEE Transactions on Software Engineering, 1994, pp. 476-493.

[29] C. Sant' ANNA, et al., On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework, In Proceeding of Brazilian Symposium. On Software Engineering, 2003, pp. 19-34.

[30] Ian Sommervile, Software Engineering, Pg. 373-375, Sixth edition, February 2000.

[31] Alessandro F. Garcia & Carlos J. P. De Leucena. An Aspect-Based Object-Oriented Model for Multi-Agent Systems. In Proceeding of the 20th Advanced Separation of Concerns Workshop at ICSE' 2001, Toronto, Canada, May 2001.

Appendix B

Publication

4th IEEE International Conference on Emerging Technologies
Rawalpindi, Pakistan, 18th - 19th October, 2008

Impact of Aspect-Orientation on the Reliability of Decentralized Multi-Agent System

Hira Tabbasum, Salma Jabeen
Department of Computer Science
International Islamic University,
Islamabad, Pakistan
{htabbasum.iiui, sjabeen.iiui}@gmail.com

H. Farooq Ahmed, Abdul Ghafoor
Department of Computer Science
NUST Institute of Information Technology,
Rawalpindi, Pakistan
{drfarooq,abdul.ghafoor}@niit.edu.pk

Abstract—**Aspect Oriented Programming (AOP) provides separation of concerns and encapsulates crosscutting concerns into separate modules called 'aspects', thereby enhancing the software quality. This paper presents the impacts of aspect-orientation on the reliability of a decentralized Multi Agent System (MAS). We compared aspect-oriented and object-oriented versions of the same application in order to explore to what extent each implementation provides a reliable system. We evaluated both versions of the system and found that the aspect-oriented design is more reliable as it has brought a loosely coupled and less complex system.**

Keywords-crosscutting concerns; aspect-orientation; decentralized multiagent systems

I. INTRODUCTION

Aspect Oriented Software Development (AOSD) [1] is a new emerging technology that provides separation of concerns in software construction [2]. Separation of concerns is a central software engineering principle that should be applied throughout the development process, from requirement to implementation [3]. It states that a given problem involves different kinds of concerns, which should be identified and separated in order to manage the complexity of the system [4]. Concerns can be classified into two categories those are core concerns and crosscutting concerns. Core concerns deal with the basic functionality of a system while crosscutting concerns span multiple modules and deal with non-functional requirements [5]. Aspect Oriented Programming (AOP) [6] provides improved modularization that encapsulates crosscutting concerns into separate modules known as 'aspects' [7].

Software engineering of multi-agent system involves the classification of concerns into two categories: agenthood concerns and additional concerns. Agenthood concerns include knowledge, interaction, adaptation, and autonomy. While additional concerns include mobility, learning and collaboration. From these mobility, interaction, learning, autonomy and collaboration are crosscutting concerns [8]. A group of researchers have worked on aspect-oriented architecture [8], modeling [9] and engineering [2] of MASs but there is no empirical evidence whether AOP helps in improving the reliability of a decentralized Multi-Agent system, thereby hindering the adaptation of AOP for such system.

This paper presents a case-study in which we have compared the reliability of aspect-oriented (AO) and object-oriented (OO) design of decentralized Agent Management System (AMS) of SAGE (Scalable, fault tolerant Agent Grooming Environment). SAGE is FIPA [10] compliant decentralized multi-agent system [11]. In SAGE, scalability and fault tolerance is achieved up to some extent through its architecture but reliability is still a major issue due to the excess of tangling and scattering of code in one of its components i.e., AMS. In addition, internal attributes like coupling and cohesion also affect system's external attributes like reliability, reusability and maintainability [12] while SAGE is a highly coupled and complex system because its code is not well optimized [13].

This paper also explains those crosscutting concerns that come across the development of a decentralized AMS and implemented those concerns with AspectJ [14]. It involves three sort of crosscutting concerns those are Knowledge Distribution, Exception Handling and Knowledge Consistency and core concerns like Agent Management and Peer Management. We have also evaluated both versions of decentralized AMS.

The rest of the paper is organized as follows. Section 2 presents the object-oriented design (OOD) of decentralized AMS of SAGE. Section 3 describes the crosscutting modules and aspect-oriented design (AOD) [15] of decentralized AMS. Section 4 explains the selected metrics to evaluate both versions. Section 5 shows the results of metrics in tabular form. Section 6 analyzes the results of metrics and section 7 has some concluding remarks.

II. DECENTRALIZED AGENT MANAGEMENT SYSTEM

SAGE [16] is following distributed architecture with decentralized management for fault tolerance of multi-agent systems. Its architecture is a blend of client/server paradigm and peer-to-peer. Instead of having the centralized location for management the ownership rights have been distributed to peer entities which are solely responsible for their roles and actions. These peer entities are part of a single Agent Platform

978-1-4244-2211-1/08/$25.00 ©2008 IEEE

and are managed by AMS. This system was ideal for our case study due to several reasons. First, it is the only multi-agent system that has a decentralized agent management platform and the behavior of Aspect-oriented programming has not yet been proved in this context. Second, scalability and reliability always remained a key attribute for multi agent systems. Finally, its realization involves a number of core and crosscutting concerns; those are of great importance in case of decentralized multi-agent systems.

A. *Object-oriented design of Agent Management System*

The object-oriented version of decentralized AMS was implemented using Java programming language. However, Java is good to deal with inheritance and polymorphism, but it may also be the cause of introducing scattering and tangling in the code. When the AMS was reverse-engineered in order to obtain object-oriented design the same condition was observed. The system was analyzed with both class diagrams and interaction diagrams. The result showed that due to high inheritance between classes the system became highly coupled. The methods were incoherently calling each other making the flow of program more complex and not understandable. Moreover, although the system has become decentralized to achieve fault tolerance and scalability, but it has become difficult to organize system components into modules according to their functionality. Therefore, there was a need to implement the system with a technology that could provide linguistics mechanisms for separate expressions of concerns in AMS and weaving these concerns with the system's primary concerns. Figure 1 shows the object-oriented class diagram of decentralized Agent Management System of SAGE.

III. MAKING AN ASPECT-ORIENTED DESIGN

Aspect orientation is the recognisation of the development of software system with respect to many concerns [17]. The development of software system with respect to concerns

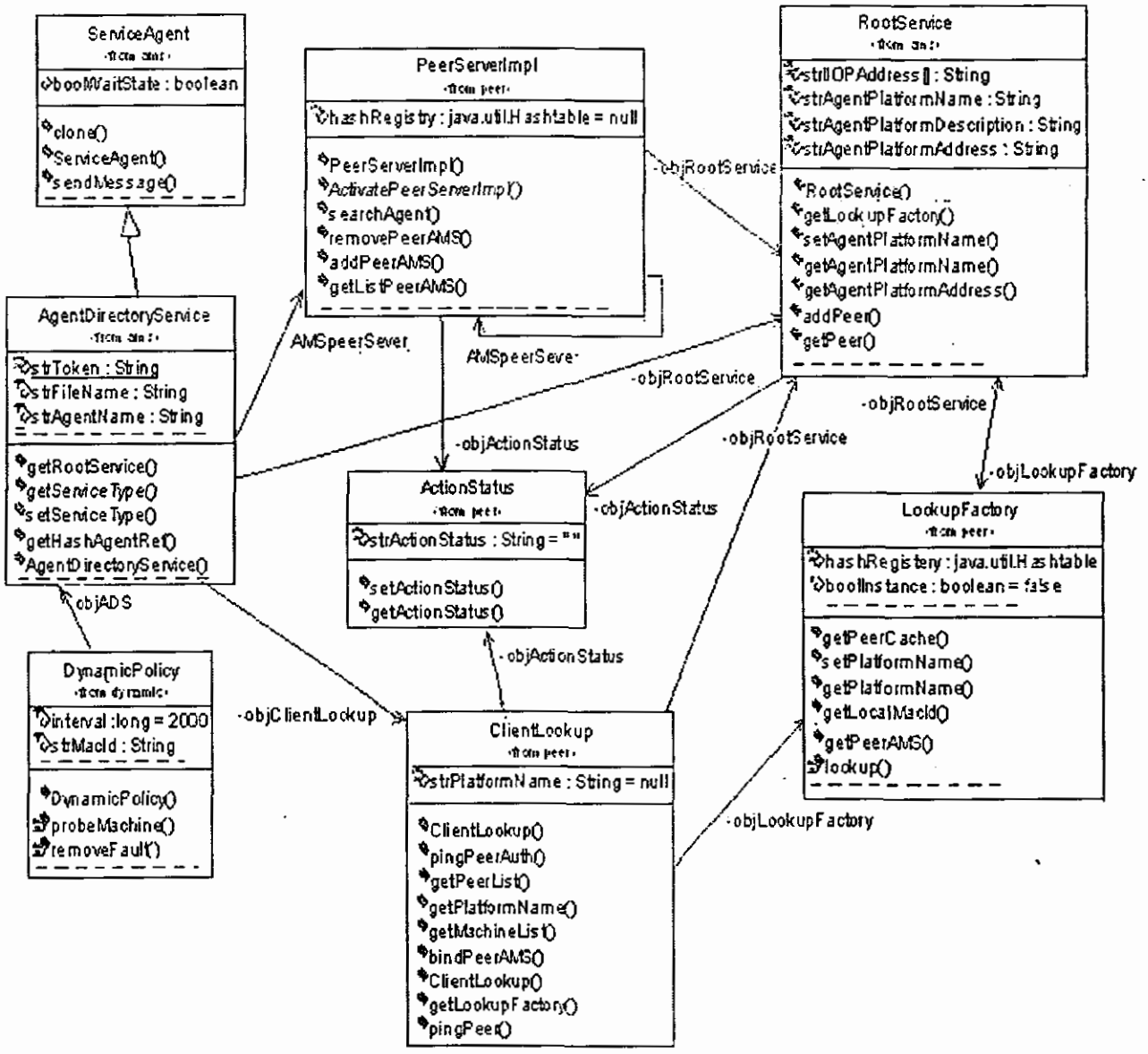

Figure 1. Object-oriented Class diagram of decentralized Agent Management System.

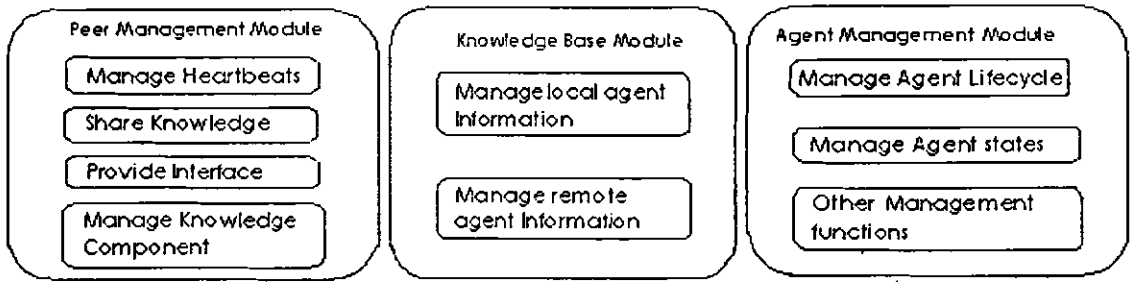

Figure 2. Peer Management, Knowledge Base and Agent Management modules of Agent Management System

could be achieved by separating classes into concerns (or modules) according to the agenthood properties they possess. The AOD [15] is developed then by identifying the crosscutting among these concerns (or modules).

A. Identification of modules in Agent Management System

Every multi-agent system has a desired set of properties [8] (or agenthood properties) and it is developed in order to achieve those properties. These properties are knowledge, mobility, learning, etc. Therefore, there is a need to go through the AMS in terms of these properties i.e., to see which part (module) of system is related to which agenthood property. It is more likely to analyze properties in a single logical AMS distributed over multiple machines. Therefore, the basic modules or concerns identified according to these properties are:

1. Peer Management
2. Knowledge Base, and
3. Agent Management

Here is the description of these modules:

1) Peer Management module: The Peer management module is responsible for managing all the operations of peer entities:

1. It manages heartbeats or liveliness of peer machines after a fixed interval of time provided it is done dynamically.
2. It provides interface to the peer entities through RMI (client/server) layer. So any request to server made by the client of peer machines will be acknowledged by the peer management module.
3. It helps peer components or entities to share knowledge with each other. For example, searching and providing information about a certain agent on a peer agent's request.
4. It also helps to manage the knowledge component in case of any sort of modification in peer agent's information.

2) Knowledge Base module: Though Knowledge base component manages local agent and remote agent information, it is also managed by Peer Management module. Its functions are:

1. To keep information about agents i.e., their state, life cycle, etc.
2. To perform functions like searching information about a particular agent, deleting, adding or modifying the agent information and providing the information about Agent Platform (AP) description.

3) Agent Management module: The responsibilities of Agent Management module are similar to those of AMS's responsibilities: register, deregister, managing life cycle and states of agents, etc.

Figure 2 represents the functions of these modules.

B. Aspect-oriented design of Agent Management System

The modules identified in previous section helped us to identify the crosscutting in the system more appropriately. The main root of crosscutting was Knowledge Base module whose management functions were spread over Peer Management and Agent Management modules. The reason was the shared registry information due to which it was difficult to maintain the distribution of information or updating the information in case of a change in registry, in a peer-to-peer paradigm. Using bidirectional RMI with a combined client/server at both ends does not help in reducing coupling. In addition, consistent registry information at all peer entities brought the issue of reliability. These issues were resolved by identifying three main aspects that were Knowledge Distribution, Knowledge Consistency and Exception Handling aspects. Exception Handling was selected as a primary crosscutting concern because there were roughly 63 instances of exception handling in 25 classes of AMS that makes its existence about 252% in the system. That is, double the number of classes of AMS. Therefore, it was necessary to choose it as a crosscutting concern in order to reduce tangling from the code. The figure 3 shows the AOD of AMS. It shows which class of a module aspect is crosscutting. For example, KnowledgeConsistencyAspect crosscuts all the classes in Peer Management, Knowledge Base and Agent Management modules. Note that classes in each module are selected on the basis of percentage of functionality of a particular module they possessed. The AO version of decentralized AMS of SAGE is implemented using AspectJ [14].

IV. THE METRICS

The reliability can be measured by measuring the internal attributes like coupling, cohesion and complexity of a system [18] [19]. Chidamber and Kemerer's (CK) metrics suite [20] are best to capture the above-mentioned attributes of a

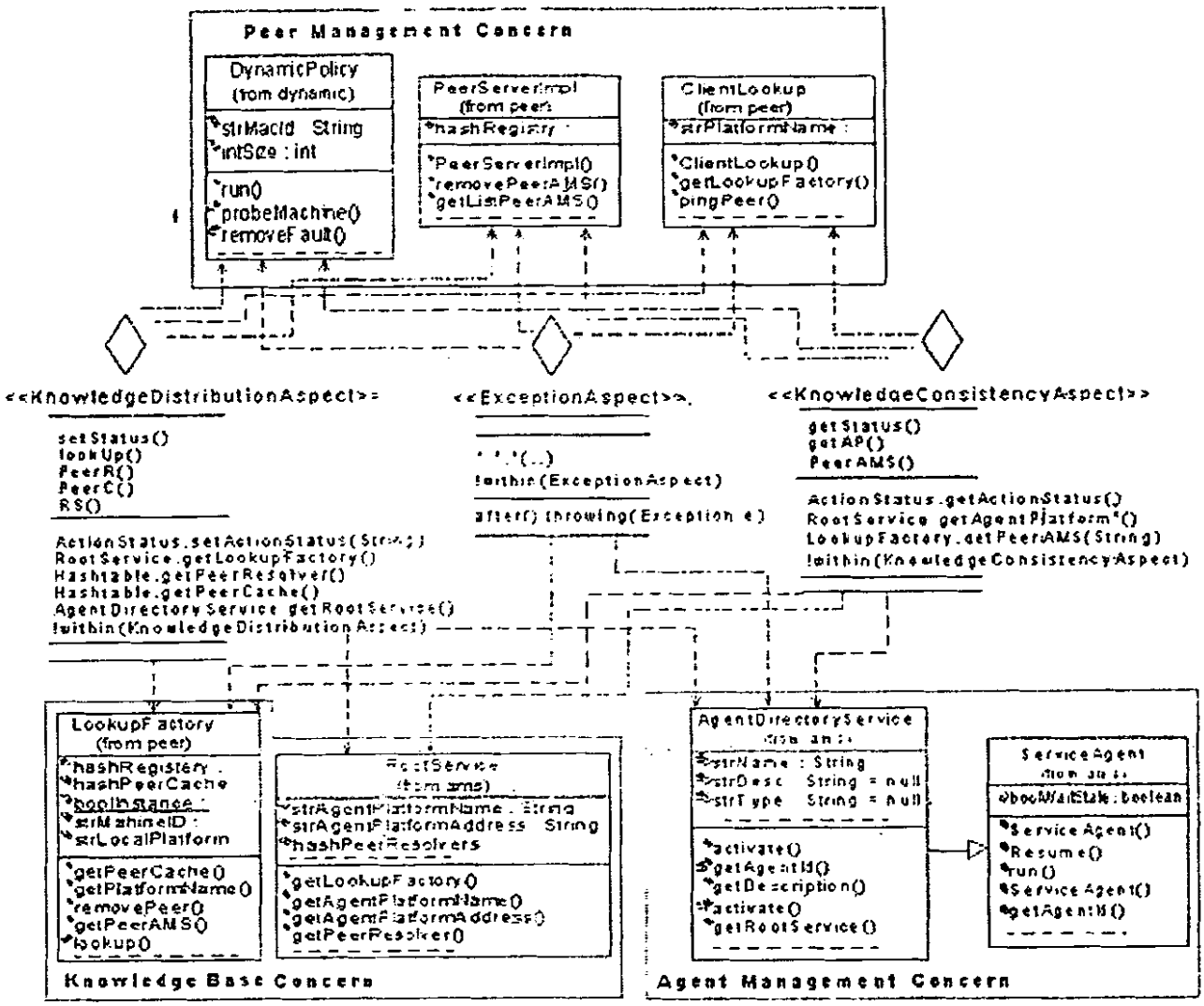

Figure 3. Crosscutting agent concerns in Peer management, Knowledge base and Agent management modules

software. Therefore, we are using a metrics suite from [21], which is based on the refinement of CK metrics for aspect-oriented development and reusing LOC metrics. These metrics capture the degree to which a single system concern maps to design components (classes and aspects) and operations (methods and advice) [7]. The chosen metrics have been applied on class diagrams of both the versions where it was necessary and behavioral diagrams were also consulted. Table 1 briefly defines each metric and associates it with the relevant software attribute. We grouped the metrics to measure a certain attribute, according to our own requirements.

V. EVALUATION

Table 2 and 3 present the computed metric values for both AO and OO versions. We compared both AO and OO systems on the basis of each metric value as follows:

Coupling

- Coupling Between Components (CBC)

From the metric values, it can be noticed that in AOD, coupling is increasing but if we observe the AOD in Figure 3, we can understand that the coupling between core classes is decreasing and the coupling between core classes and aspects is increasing.

- Depth of Inheritance Tree (DIT)

Only one class with the name of ServiceAgent has a subclass a AgentDirectoryService which does not pay a remarkable effect on a system.

Cohesion

- Lack of Cohesion in Operations (LCOO)

Values for the LCOO decreased in aspect-oriented design, which means the AO version of the system is more cohesive as compared to OO version. System with more cohesion is more reliable and efficient.

Complexity

- Weighted Operations per Component (WOC)

It can be seen from the results of metrics, that in AO version, the number of operations per class/aspect is reduced

as compared to OO version. AOSD helps in decreasing the crosscutting between the classes and reduces the number of tangled methods in the class. Therefore, it decreases the overall complexity of a system.

- Lines of Code (LOC)

LOC is 1685 in the OO implementation and 1486 in AO implementation. This shows that OO system is more complex in terms of LOC.

TABLE I. THE METRIC SUITE

Attribute	Metrics	Definition
Coupling	*Coupling Between Components (CBC)*	Counts the number of other classes and aspects to which a class or an aspect is coupled.
	Depth of Inheritance Tree (DIT)	Counts how far down in the inheritance hierarchy a class or aspect is declared.
Cohesion	*Lack of Cohesion in Operations (LCOO)*	Measures the lack of cohesion of a class or an aspect in terms of the amount of method and advice pairs that do not assess the same instance variable.
Complexity	*Weighted Operations per Component (WOC)*	Counts the number of methods and advice of each class or aspects or the number of its parameters.
	Lines of Code (LOC)	Counts the lines of code.
	Number of Attributes (NOA)	Counts the number of attributes of each class or aspect.

TABLE II. METRICS OBTAINED FOR OO DESIGN

METRICS		COUPLING		COHESION	COMPLEXITY		
		CBC	*DIT*	*LCOO*	*WOC*	*LOC*	*NOA*
Class Name	*DynamicPolicy*	1	0	6	5	145	7
	AgentDirectoryService	9	0	12	10	376	14
	PeerServerImp	3	0	11	14	227	1
	RootService	8	0	22	35	347	8
	ClientLookup	3	0	5	11	159	1
	LookupFactory	2	0	15	13	299	5
	ServiceAgent	0	1	4	9	132	1

TABLE III. METRICS OBTAINED FOR AO DESIGN

METRICS		COUPLING		COHESION	COMPLEXITY		
		CBC	DIT	LCOO	WOC	LOC	NOA
Class Name	DynamicPolicy	3	0	2	3	133	5
	AgentDirectoryService	3	0	9	6	302	11
	PeerServerImp	3	0	5	9	192	1
	RootService	2	0	13	22	258	5
	ClientLookup	3	0	2	7	133	1
	LookupFactory	3	0	8	11	231	5
	ServiceAgent	0	1	9	13	130	1
	KnowledgeDistributionAspect	6	0	5	6	48	5
	ExceptionAspect	5	0	3	1	22	3
	KnowledgeConsistancyAspect	6	0	7	4	37	3

- Number of Attributes (NOA)

It can be noted, that object-oriented version is more complex in terms of NOA, while NOA is reduced in aspect-oriented version.

VI. RESULTS

Coupling, cohesion and complexity plays a vital role in the reliability of a system at design level. As we can see from the computed metrics, AOD of a decentralized AMS resulted into a loosely coupled, more cohesive and less complex system. Coupling between core and aspectual classes increased in AOD but other metrics showed good results for aspect-oriented version of the system. SAGE has a scalable and fault tolerant decentralized architecture but realization is equally important to build a fault tolerant and reliable system. Our experimentation resulted into a positive impact of aspect-orientation on the reliability of a decentralized AMS.

VII. CONCLUSIONS

This paper presented AO and OO implementations of decentralized agent-management system of SAGE. In this study, we also mentioned the crosscutting concerns of decentralized AMS. Through this analysis, we found that AOP is superior to object orientation at design and implementation level. AOP helps in improving software internal attributes and leads to better values for software external attributes such as reliability. AOP is equally helpful in achieving a quality decentralized MAS as it is helpful for other applications. Therefore, software engineers should not be hesitant in using this new technique of implementation for decentralized MASs. Our experiment resulted into a good impact of Aspect-orientation on the reliability of decentralized MAS.

REFERENCES

[1] T. Elard, R. Filman, and A. Bader (eds.), "Theme section on Aspect-Oriented Programming", CACM, 44(10), 2001.

[2] Garcia, A., Silva, V., et al, "Engineering Multi-agent systems with Aspects and Patterns", Journal of the Brazilian Computer Society, July 2002, V. 8, no. 1, pp. 57-72.

[3] Araujo, J., A. Moreira, et al, "Aspect-Oriented requirements with UML", Workshop on Aspect-Oriented Modeling with UML, 2002.

[4] Iris Groher and Thomas Baumgarth, "Aspect-Orientation from design to code", In Proceeding of Workshop on Early Aspects, Aspect-Oriented Requirements Engineering and Architecture Design; AOSD, March 2004.

[5] Ramnivas Laddad, AspectJ in action, Pages (7-11), Oreilly & Associates Inc., 2003.

[6] G. Kiczales, et al., "Aspect-Oriented Programming", In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer Verlag LNCS 1241, June 1997.

[7] U. Kulesza, et al., "Quantifying the effects of Aspect-Oriented Programming: A maintenance study", In Proceedings of the 9th International Conference on Software Reuse (ICSM'06), Philadelphia, USA, September 2006.

[8] A. Garcia, et al., " Aspectizing Multi-agent systems: from architecture to implementation", PUC-Rio, Computer Science Department, LES, SoC+Agents Group, Rio de Janeiro, RJ, Brazil, 2004.

[9] Garcia, A., Chavez, C., Choren, R, "An Aspect-Oriented modeling framework for designing Multi-agent systems", 7th Workshop on Agent-Oriented Software Engineering, AAMAS'06, Hakodate Japan, May 2006.

[10] Foundation for Intelligent Physical Agents. http://www.fipa.org

[11] Abdul Ghafoor, et al., "SAGE: next generation Multi-agent system", I Proceedings of the 2004 International Conference on Parallel an Distributed Processing Techniques and Applications pp.139-14 Vol.1.PDPTA, Navada (USA), June 2004.

[12] J. Zhao, "Measuring coupling in Aspect-Oriented systems", Technic Report, SE-142-6, Information Processing society of Japan (IPSJ), Jun 2003.

[13] A. Ghafoor, A. Shibli and H. Farooq Ahmad, "SAGE, open sourc Fault Tolerant architecture: enhancement, refactoring and debugging 21st Assurance System Symposium, Hiroshima City Universit Hiroshima, Japan 2007.

[14] Kiczales, G., Hilsdale, E., Hugunin et al., "An overview of AspectJ", I Proceedings of ECOOP 2001, Lecture Notes in Computer Science, Vo 2072, Springer (2001) 327-353.

[15] C. Von Flach and Carlos J.P., "Design level support for Aspect-Oriente development", Workshop on Advanced Separation of Concerns i Object-Oriented Systems (ASOC) at OOPSLA' 2001, Tampa Ba Florida, USA, October 14, 2001.

[16] Salman Shahid, "Agent Management System (AMS) for FIPA compliar Multi-agent system", Distributed Computing Group NUST Institute c Information Technology, Rawalpindi, 2000.

[17] R. E. Filman, et al (Eds.) Aspect-Oriented software developmen Addison-Wesley, 2005.

[18] Dr. Linda Rosenberg, et al, "Software Metrics and Reliability", 9t International Symposium on Software Reliability Engineering Germany Nov 1998.

[19] Jubair J. Al-ja'afer, Khair Eddin M. Sabri, "Chidamber-Kemerer(CK and Lorenze-Kidd(LK) Metrics to assess Java programs", In Proceedin of International Workshop on Software Systems (IWSS), 2004.

[20] Chidamber, S. and Kemerer, C., "A metric suite for Object Oriente design" IEEE Transactions on Software Engineering 20(6)(1994) 47 493.

[21] C. Sant' ANNA, et al., "On the reuse and maintenance of Aspec Oriented software: An assessment framework", Proc. Brazilia Symposium. On Software Engineering, 2003, 19-34.

Appendix C

Interfaces

C.1 VMA

C 1.1 Visual Manager GUI

The graphical user interface of *AMS Visual Manager* is shown below:

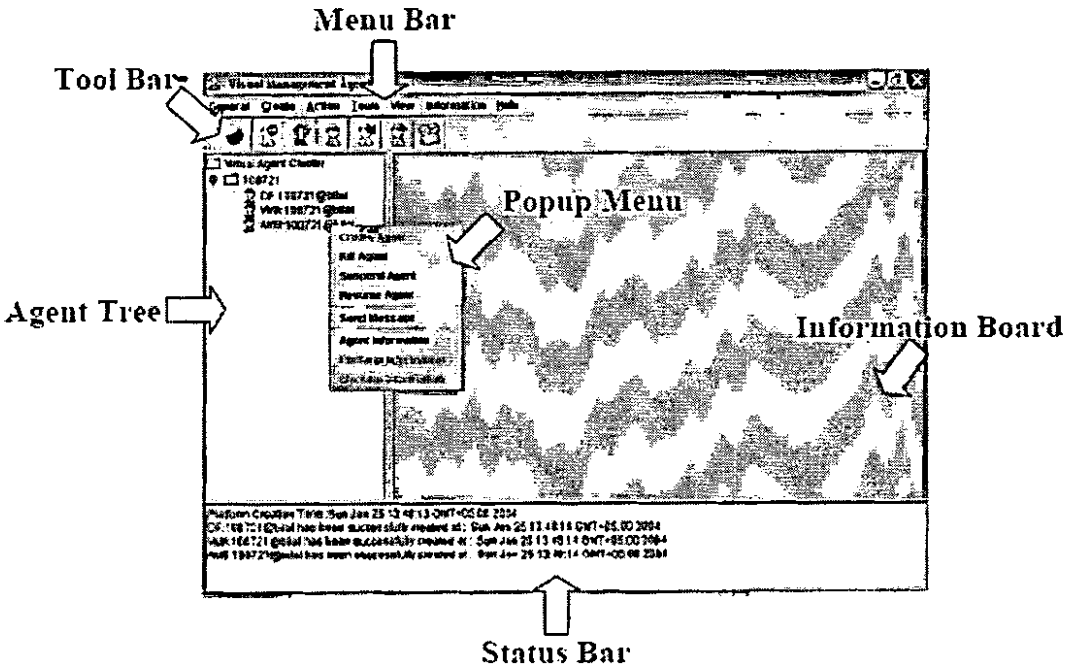

Figure 1.1 AMS Visual Manager GUI

The graphical user interface is composed of six main components. These are:

- Agent Tree.
- Menu Bar.
- Tool Bar.
- Popup Menu.
- Status Bar.
- Information Board.

Using these components user can interact with the agent and can perform multiple actions which are discussed later in detail.

1.2 Component Details

Agent Tree

Agent Tree is one of the most important components of *AMS Visual Manager*. *Agent Tree* is a three level hierarchic tree.

- First level .....Cluster Name *(Root Node)*
- Second level .....Machine Name
- Third level .....Agent Name *(Leaf Node)*

The first level node or the root node shows the name of the cluster. The second level node represents the machine on which the agents are created and registered. Third level node or the leaf node represents the agents that are registered with the machine under which they appear.

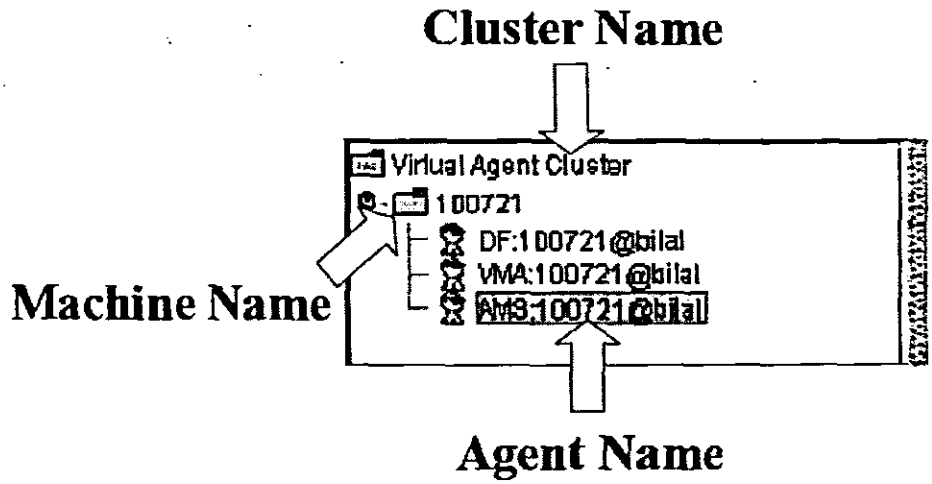

Figure 1.2 AMS Visual Manager's AgentTree

Whenever the *AMS Visual Manager* is started the *Agent Tree* is populated with the machines comprising the platform and the agents residing on those machines.

Machine Name

The machine name is represented in terms of its IP address in reverse order.

Agent Name

Agent name is composed of three segments

1. The agent name specified by the user
2. Reverse IP address of the machine on which the agent is to be created
3. Platform name

Full Agent Name = <agent name> : <reverse IP address> @ <platform name>

e.g.*VMA:100721@niit206*

where Agent name = *VMA (First segment)*

Reverse IP address = *100721 (Second segment)*

Platform name = *niit206 (Third segment)*

Menu Bar

The *Menu Bar* is a very flexible component that adds to the user friendliness of any graphical user interface. Using the *Menu Bar* the user can perform multiple actions. *AMS Visual Manager* also provides *Menu Bar* to its users. It consists of seven *Menus* that are:

- General
- Create
- Action
- Tools
- View
- Information
- Help

General

The General *Menu* consists of two *Menu Items*

- **Close AMS Visual Manager**

By using this option the user can close the *AMS Visual Manager*. When the user selects this option, the user is asked for confirmation to close the *AMS Visual Manager*. After confirmation a message is created by the *AMS Visual Manager* for *AMS* specifying to kill the *AMS Visual Manager*. After the creation the *ACL* message is sent to *AMS* via *MTS*. Then the reply message is created and sent to *AMS Visual Manager* by *AMS* via *MTS*.

After receiving the reply message from *AMS*, *AMS Visual Manager* closes its graphical user interface without causing any effect on the platform.

• Shutdown Platform

By using this option the user can close the entire platform. When the user selects this option, the user is asked for confirmation to close the platform. After confirmation a message is created by *AMS Visual Manager* for *AMS* specifying to close the platform. After creation, the *ACL* message is sent to *AMS* via *MTS*. On receiving this message *AMS* kills all the agents that are active at that time and then finally closes the application.

Create

The Create *Menu* consists of two *Menu Items*

• DF Visual Manager

By using this option the user can launch the *DF Visual Manager* which provides a graphical user interface for the management of *Directory Facilitator (DF)*. When the user selects this option, a message is created by the *AMS Visual Manager* for *AMS* specifying to create the *DF Visual Manager* agent. After creation, the *ACL* message is sent to *AMS* via *MTS*. On receiving this message *AMS* will create *DF Visual Manager* agent. The user will be shown the graphical user interface to interact with *Directory Facilitator*. Then the reply message is created and sent to *AMS Visual Manager* by *AMS* via *MTS*. After receiving the reply message from *AMS*, *AMS Visual Manager* updates its *Agent Tree* view by adding the entry of *DF Visual Manager* agent in the *Agent Tree*.

• TestAgent

By using this option the user can launch the *TestAgent*. When the user selects this option, a message is created by the *AMS Visual Manager* for *AMS* specifying to create the *TestAgent*. After creation, the *ACL* message is sent to *AMS* via *MTS*. On receiving this message *AMS* will create *TestAgent* and its graphical user interface will be shown to the user. Then the reply message is created and sent to *AMS Visual Manager* by *AMS* via *MTS*. After receiving thereply message from *AMS*, *AMS Visual Manager* updates its *Agent Tree* view by adding the entry of *TestAgent* in the *Agent Tree*.

Action

The Action *Menu* consists of five *Menu Items*

• Create Agent

By using this option the user can create a new agent. First the user has to specify the machine on which the user wants to create the agent. After that when the user selects this option, the user will be asked to provide the following information

- Agent name
- Class Path
- Arguments

After getting all the necessary information from the user a message is created by the *AMS Visual Manager* for *AMS* specifying to create a new agent with the name specified. After creation, the *ACL* message is sent to *AMS* via *MTS*. On receiving this message *AMS* will create a new agent with the user specified name. Then the reply message is created and sent to *AMS Visual Manager* by *AMS* via *MTS*. After receiving the reply message from *AMS*, *AMS Visual Manager* updates its *Agent Tree* view by adding the entry of the agent created in the *Agent Tree*.

• Kill Agent

By using this option the user can kill the specified agent. First the user has to select a particular agent from the *Agent Tree*. After that when the user selects this option, the user is asked for confirmation to kill the specified agent. After confirmation a message is created by *AMS Visual Manager* for *AMS* specifying to kill the specified agent. After creation, the *ACL* message is sent to *AMS* via *MTS*. On receiving this message *AMS* will kill the agent as specified by user. Then the reply message is created and sent to *AMS Visual Manager* by *AMS* via *MTS*. After receiving the reply message from *AMS*, *AMS Visual Manager* updates its Agent Tree view by eliminating the entry of the agent in the Agent Tree.

• Suspend Agent

By using this option the user can suspend the specified agent. First the user has to select a particular agent from the *Agent Tree*. After that when the user selects this option, the user is asked for confirmation to suspend the specified agent. After confirmation a message is created by *AMS Visual Manager* for *AMS* specifying to suspend the specified agent. After creation, the *ACL* message is sent to *AMS* via *MTS*. On receiving this message *AMS* will suspend the agent as specified by the user. Then the reply message is created and sent to *AMS Visual Manager* by *AMS* via *MTS*. After receiving the reply message from *AMS*, *AMS Visual Manager* notifies the user by showing the confirmation message in the *Status Bar* that the agent has been suspended.

• Resume Agent

By using this option the user can resume the specified suspended agent. First the user has to select a particular suspended agent from the *Agent Tree*. After that when the user selects this option, the user is asked for confirmation to resume the agent. After confirmation a message is created by *AMS Visual Manager* for *AMS* specifying to resume the specified agent. After creation, the *ACL* message is sent to *AMS* via *MTS*. On receiving this message *AMS* will resume the agent as specified by the user. Then the reply message is created.
and sent to *AMS Visual Manager* by *AMS* via *MTS*. After receiving the reply message from *AMS*, *AMS Visual Manager* notifies the user by showing the confirmation message in the *Status Bar* that the agent has been resumed.

• Send Message

By using this option the user can send the messages to other specified agents. First the user has to select a particular agent from the *Agent Tree* as the sender of the message. After that, when the user selects this option the user will be shown the *message creation window* through which the user can compose and send message(s). The window consists of four fields which are:

1. Sender Name

When the ***message creation window*** starts up this field is already populated with the name of sender agent.

2. Receiver Name

In this field the user has to specify the name of the recipient agent. This field has two buttons attached with it which are:

• Add

By using this button a popup window is appeared showing the names of the agents currently residing in the platform. User then selects the desired recipient of the message. The recipient name is then added to the Receiver field.

• Remove

By using this button the user can delete the selected recipient name from the Receiver field. The user can specify multiple receivers for the message.

3. Communicative Act

Using this field the user can specify the type of communicative act for the message to be sent to the specified recipient(s).

4. Contents

Using this field the user can specify the message contents. After specifying the inputs, the user submits the request. After submission a message is created by *AMS Visual Manager* for *MTS* specifying to send the message to the specified recipient(s). After creation, the *ACL* message is sent to *MTS* which is responsible for forwarding the message to the desired recipient(s).

Tools

The Tools *Menu* consists of three *Menu Items*

• Save Log

By using this option the user can save the contents of the *Status Bar*. When the user selects this option, a *save* file dialog appears in which the user has to supply the name and location of the file to be saved.

• Refresh Status Bar

By using this option the user can refresh the *Status Bar*. When the user selects this option, any contents in the *Status Bar* are cleared.

• Change Status Bar Color

By using this option the user can change the color of the text shown in the *Status Bar*. When the user selects this option, a *Color chooser* dialog appears in which the user selects a particular color. After color selection the color of the text in the *Status Bar* is updated.

View

The View *Menu* consists of two checkbox *Menu Items*

• View Status Bar

By checking the *Menu Item* the user can view the *Status Bar*. If the user doesn't want to view the *Status Bar,* the user can uncheck the *Menu Item.*

• View Agent Tree

By checking the *Menu Item* the user can view the *Agent Tree*. If the user doesn't want to view the *Agent Tree,* the user can uncheck the *Menu Item.*

Information

The Information *Menu* consists of three *Menu Items*

• Platform Information

By using this option the user can view the information about the agent platform. When the user selects this option the user will be shown the *platform information window* through which the user can view the desired information that includes:

1. Platform creation time

It shows the time at which the platform was started.

2. Platform name

It shows the name of the platform.

3. Total machines in cluster

It shows the total number of machines in the cluster that form the agent platform.

4. Total agents created

It shows the total number of agents that have been created since the agent platform was created.

5. Number of active agents

It shows the total number of agents that are currently active in the agent platform.

6. Total agents killed

It shows the total number of agents that have been killed since the agent platform was created.

7. Number of suspended agents

It shows the total number of agents that are currently suspended in the agent platform.

• Machine Information

By using this option the user can view the information about a particular machine. First the user has to select a particular machine from the *Agent Tree*. After that when the user selects this option the user will be shown the *machine information window* through which the user can view the desired information that includes:

1. Machine start time

It shows the time at which the machine was started and made the part of the cluster.

2. Machine name

It shows the name of the particular machine.

3. JDK Version

It shows the version of Java Virtual Machine *(JVM)* running on that particular machine.

4. OS Version

It shows the version of Operating System (OS) running on thatparticular machine.

5. Total agents created

It shows the total number of agents that have been created since the machine was started.

6. Number of active agents

It shows the total number of agents that are currently active on that particular machine.

7. Total agents killed

It shows the total number of agents that have been killed since the machine was created.

8. Number of suspended agents

It shows the total number of agents that are currently suspended on that particular machine.

• Agent Information

By using this option the user can view the information about a particular agent. First the user has to select a particular agent from the *Agent Tree*. After that when the user selects this option the user will be shown the *agent information window* through which the user can view the desired information that includes:

1. Agent Name

It shows the selected agent name.

2. Agent Owner

It shows the owner of the selected agent.

3. Agent State

It shows the state of the selected agent. An agent can be in one of 13 states that are:

- Active
- Suspend
- Transit
- Unknown
- Create
- Invoke
- Destroy
- Quit
- Resume

- Wait
- Wakeup
- Move
- Execute

Help

The help ***Menu*** consists of one ***Menu Item***

- **About**

By using this option the user can view the details about ***AMS Visual Manager***.

Tool Bar

Tool Bar is a very flexible component that adds to the user friendliness of any graphical user interface. Using the ***Tool Bar*** the user can perform multiple actions. *AMS Visual Manager* also provides *Tool Bar* to its users. *As* discussed in above section, the following actions can be performed using the *Tool Bar*

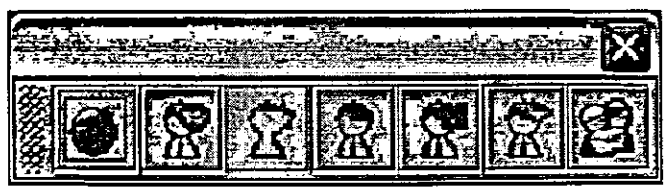

Fig. 1.3 Visual Managrer's Too Bar

1. Shutdown Platform
2. Close AMS Visual Manager
3. Create new agent
4. Kill Agent
5. Suspend agent
6. Resume agent
7. Send Message to other agents

Popup Menu

AMS Visual Manager provides *Popup Menu* to its users. Following actions can be performed using the *Popup Menu.*

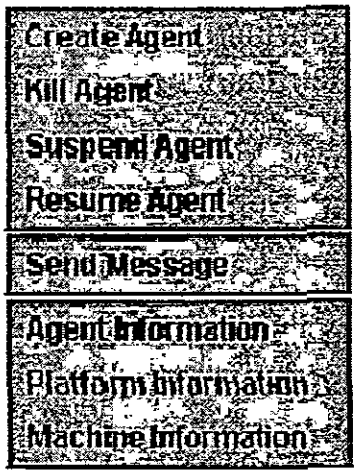

Fig. 1.4 Visual Manager's Pop Up Menu

1. Create new agent
2. Kill Agent
3. Suspend agent
4. Resume agent
5. Send Message to other agents
6. View Platform information
7. View Machine information
8. View agent Information

The *PopupMenu* can be launched by selecting any node in the *Agent Tree* and then pressing the right mouse button.

Status Bar

This component is used to inform the user about the current activities taking place in the agent platform.

Information Board

This component shows the information windows to the user on request that includes:

- Platform information.
- Machine information.
- Agent information.

Appendix D

Coding

D.1 DynamicPolicy

This is class is from Peer Management concern. This class is responsible for the dynamic probing (peer to peer). It checks the status of the other machines on the agent platform and in case of failure it removes the peer machine from the platform.

```
package ams.probe.dynamic;

import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Hashtable;

import acl.ACLMessage;
import acl.ACLPerformatives;
import acl.CFPredicate;
import acl.aclcodec.ACLCodec;
import acl.ontology.management.ManagementOntology;
import acl.sl.codec.SLTokenizer;
import ams.AgentDirectoryService;
import ams.AgentId;
import ams.AgentStates;
import ams.RSFactory;
import ams.Utility;
import ams.peer.ActionStatus;
import ams.peer.PeerAMS;

public class DynamicPolicy extends Thread {

        AgentDirectoryService objADS;
        long interval = 2000;
        PeerAMS peerAMS;
        String strMacId;
        Enumeration enum;
        int intSize;

        /**
        * Constructor for the DynamicPolicy class
        * @param AgentDirectoryService
        */
        public DynamicPolicy(AgentDirectoryService objADS) {
                this.objADS = objADS;
                this.start();
        }

        public void run() {
                while (true) {
                        try {
                                sleep(interval);
                                enum = objADS.getRootService().getPeerResolver().keys();
                                this.probeMachine();
                        } catch (Exception e) {
                                e.printStackTrace();
```

```
                }
            }
        }

        private void probeMachine() {
            try {
                String strLocalAMS;
                Utility objUtility = new Utility();
                strLocalAMS =
objUtility.setMacID(java.net.InetAddress.getLocalHost().getHostAddress());
                ArrayList alMachineList = objADS.getRootService().getMachineList();

                if (intSize != alMachineList.size()) {
                    for (int i = 0; i < alMachineList.size(); i++) {
                        try {
                            peerAMS =
objADS.getRootService().getLookupFactory().getPeerAMS(alMachineList.get(i).toString());
                            peerAMS.updateMachineList(alMachineList);
                            System.out.println("-------------------------
Machine Information Updated-------------------------");

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

                int index = alMachineList.indexOf(strLocalAMS);
                if (index == alMachineList.size() - 1) {
                    index = 0;
                } else
                    index = index + 1;

                ActionStatus objActionStatus;

                boolean boolCheck = false;
                strMacId = (String) alMachineList.get(index);

                peerAMS =
objADS.getRootService().getLookupFactory().getPeerAMS(strMacId);
                if (peerAMS == null) {
                    System.out.println("Peer AMS is null");
                }
                if (!strMacId.equals(strLocalAMS)) {
                    boolCheck = peerAMS.pingResponse();
                    if (boolCheck == true) {
                        for (int i = 0; i <
objADS.getRootService().getMachineList().size(); i++) {

        System.out.println(objADS.getRootService().getMachineList().get(i));
                        }
                    } else
                        System.out.println("Status is not OK " + strMacId);
                }
```

```
                    intSize = objADS.getRootService().getMachineList().size();
                } // end try

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

        private void removeFault(String strMacId) {
                ArrayList alMachineList;
                Hashtable hashPeerResolvers = null;
                System.out.println("----------I m in the method to handle the error----------
" + strMacId);
                hashPeerResolvers = objADS.getRootService().getPeerResolver();
                hashPeerResolvers.remove(strMacId);
                objADS.getRootService().getMachineList().remove(strMacId);

        objADS.getRootService().getLookupFactory().getPeerCache().remove(strMacId);
                System.out.println("Removed entry from local machine");
                alMachineList = objADS.getRootService().getMachineList();
                Enumeration e = hashPeerResolvers.keys();
                for (int i = 0; i < alMachineList.size(); i++) {
                    try {
                        peerAMS =
objADS.getRootService().getLookupFactory().getPeerAMS(alMachineList.get(i).toString());
                        peerAMS.updateMachineList(alMachineList);
                        peerAMS.removeMachine(strMacId);
                        System.out.println("******////////*********Removed from
machine" + e.toString());
                        sendMsgVMA(strMacId);
                    } catch (Exception ex) {
                        ex.printStackTrace();
                    }

                } // end while

        } // end method

        private void sendMsgVMA(String strMacId) {
                ArrayList alMachineList = objADS.getRootService().getMachineList();
                Hashtable hashPeerResolvers = null;

                System.out.println("----------I m in the method to handle the error----------
" + strMacId);
                hashPeerResolvers = objADS.getRootService().getPeerResolver();
                Enumeration enum = hashPeerResolvers.keys();

                ACLMessage objACLMessage = new
ACLMessage(ACLPerformatives.INFORM);
                ACLMessage objACLMessage1 = new
ACLMessage(ACLPerformatives.INFORM);

                objACLMessage.setOntology(ManagementOntology.NAME);
```

```
                objACLMessage.setSender(objADS.getAgentId());

                objACLMessage1.setOntology(ManagementOntology.NAME);
                objACLMessage1.setSender(objADS.getAgentId());
                try {

                        while (enum.hasMoreElements()) {

                                objACLMessage.addReceiver(new AgentId("VMA:" +
enum.nextElement().toString() + "@" + RSFactory.getPlatformName(), null, null, null,
AgentStates.ACTIVE));
                                objACLMessage1.addReceiver(new AgentId("VMA:" +
enum.nextElement().toString() + "@" + RSFactory.getPlatformName(), null, null, null,
AgentStates.ACTIVE));

                        }
                        CFPredicate objFailure = new
CFPredicate(ManagementOntology.FAILUREMACHINE);
                        objFailure.set(ManagementOntology.FAILUREMACHINE_NAME,
strMacId);
                        SLTokenizer objSLTokenizer = new
SLTokenizer(ManagementOntology.getInstance());

                        CFPredicate objFailure1 = new
CFPredicate(ManagementOntology.FAILUREMACHINE);
                        objFailure1.set(ManagementOntology.FAILUREMACHINE_NAME,
strMacId);
                        SLTokenizer objSLTokenizer1 = new
SLTokenizer(ManagementOntology.getInstance());

                        objACLMessage.setContent(objSLTokenizer.encode(objFailure));

                        objACLMessage1.setContent(objSLTokenizer.encode(objFailure1));

        objACLMessage.setX_AuthenticationId(AgentDirectoryService.getFirstTok());
                        objADS.sendMessage(objACLMessage);
                        ACLCodec aclCodec = new ACLCodec();
                        objADS.sendMessage(objACLMessage1);
                        System.out.println(aclCodec.encode(objACLMessage));
                        System.out.println(aclCodec.encode(objACLMessage1));
                } // END TRY

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

}
```

D.2 PeerServerImpl

This is class is also part of Peer Management concern. This is the server class of AMS RMI.

```
package ams.peer;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Hashtable;

import ams.AgentId;
import ams.AgentRegistry;
import ams.RootService;

 public class PeerServerImpl extends UnicastRemoteObject implements PeerAMS
 {
 private RootService objRootService = null;
 private Hashtable hashRegistry = null;
 private ActionStatus objActionStatus = new ActionStatus();

    AgentId objAgentId = new AgentId();
    PeerServerImpl AMSpeerSever;
 /**
  * Constructor for the PeerServerImpl Class
  * @throws RemoteException
  */
 public PeerServerImpl() throws RemoteException
 {
                super();
     System.out.println("sdfsdfsdfsdfsdfsdfsdfdsf");
 }


 /**
  This method is used to set RootService and shared registry Information
  @param objRootService
  @param hashRegistry
  @throws RemoteException
  */
 public void ActivatePeerServerImpl(RootService objRootService, Hashtable
hashRegistry)throws RemoteException
 {

  if (objRootService != null)
     this.objRootService = objRootService;
```

```
  if (hashRegistry != null)
     this.hashRegistry = hashRegistry;

 }

 /**
  This method returns true if the pingResponse is successfull
  @return
   @throws RemoteException
  */
 public boolean pingResponse() throws RemoteException
 {
     return true;
 }
 /**
   This method accepts the name of the component , checks if its working and
   returns the actionStatus of the ping.
   @param strPeerComponent
   @return
   @throws RemoteException
  */
 public ActionStatus pingResponse(String strPeerComponent) throws RemoteException
 {
   try
   {
System.out.println("*****************"+strPeerComponent+"******************");
      if(strPeerComponent.equals("MTS"))
      {
         System.out.println("GOT PEERMTS");
       }

      else if(strPeerComponent.equals("peerAMS"))
      {
        System.out.println("GOT PEERAMS");
              System.out.println("AMS REBINDED");
      }

      else if(strPeerComponent.equals("peerDF"))
      {
        System.out.println("GOT PEERDF");
      }

      else if(strPeerComponent.equals("peerVMA"))
      {
        System.out.println("GOT PEERVMA");
      }
    }

    catch(Exception e)
```

```
{
    e.printStackTrace();
}
    System.out.println("In the Server Method");
    //now here we have to rebind the Specific Component
    //and send action status back to the sender that the machine is rebined now.
    objActionStatus.setActionStatus("Rebinded");
    return objActionStatus;
}

/**
 AMS is responsible for the searching of an Agent within the Platform.
 Remote search should be performed by AMS rather than agent by itself.
 @param agname
 @return
 @throws RemoteException
 */
public ActionStatus searchAgent(String strAgentName) throws RemoteException
{
 System.out.println("Inside Search Agent on PeerServerImpl");
 if (strAgentName != null)
 {
   if(hashRegistry.containsKey(strAgentName))
     objActionStatus.setActionStatus("EXISTS");
   else
     objActionStatus.setActionStatus("DOESNOTEXIST");
 }
 else
     objActionStatus.setActionStatus("NAMENULL");

 System.out.println("Inside Search Agent on PeerServerImpl Before Return");
 return objActionStatus;
}
/**
 AMS is responsible for the searching of agent description either on
 local machine or on distributed machines.
 @param agname
 @return
 @throws RemoteException
 */
public AgentId searchAgentDesc(String strAgentName) throws RemoteException
{
 AgentRegistry agentReg = null;

 if (strAgentName != null)
 {
   if (hashRegistry.containsKey(strAgentName))
   {
        agentReg = (AgentRegistry) hashRegistry.get(strAgentName);
        objAgentId = agentReg.getAgentId();
```

```
            return objAgentId;
        }
        else
        {
          System.out.println("Inside Search Agent on PeerServerImpl Returning 2 : " +
objAgentId.getAgentName());
          return null;
        }

      }
      else
      {
        System.out.println("Inside Search Agent on PeerServerImpl Before FInal Return");
        return null;
      }
    }
    /**     Name of faulty detected machines must be conveyed to other peer machines.
     @param strMacId
     @return
     @throws RemoteException
     */

    public ActionStatus removePeerAMS(String strMacId) throws RemoteException
    {
      if (strMacId != null)
      {
       if(objRootService.getPeerResolver().containsKey(strMacId))
        {
          System.out.println("Server Side PeerRemoved form :
"+objRootService.getAgentPlatformAddress()+"  :  "+ strMacId);

objActionStatus.setActionStatus(objRootService.removePeer(strMacId).getActionStatus(
));
        }
        else
          objActionStatus.setActionStatus("DOESNOTEXISTS");
      }
      else
        objActionStatus.setActionStatus("NULLNAME");
      return objActionStatus;
    }

    /**
     To add another PeerAMS to local peerResolver
     @return
     @throws RemoteException
     */
    public ActionStatus addPeerAMS(String strMacId, MainAddress objMainAddress)
throws RemoteException
    {
```

```
    if ((strMacId != null) && (objMainAddress != null))
    {
      System.out.println("Server Side Adding the PeerAMS : " + strMacId);
      objActionStatus = objRootService.addPeer(strMacId,objMainAddress);
    }
    else
      objActionStatus.setActionStatus("NULLNAME");
    return objActionStatus;
  }

  /**
   This method accepts the machineId of the machine and removes it from the local
machine
    @param MacId
    @throws RemoteException
   */
  public void removeMachine(String MacId) throws RemoteException
  {
     System.out.println("I am in the peer machine server method remove machine");
     objRootService.getPeerResolver().remove(MacId);
     objRootService.getLookupFactory().getPeerCache().remove(MacId);
  }

   /** Platform Name must be same among all the peer machines.
    @return
    @throws RemoteException
   */

  public String getPlatformName() throws RemoteException
  {
    System.out.println("Server Side: Call recieved from remote in Get Platform Name");
    if(objRootService.getAgentPlatformName()==null)
    {
     System.out.println("Agent Platform name is null");
    }
    return objRootService.getAgentPlatformName();
  }
  /**
   ACC Platform Address returned
   @return
   @throws RemoteException
   */
  //@TODO new change mobility critical method not found
   public String[] getACCPlatformAddress() throws RemoteException
   {
    if(objRootService.getIIOPAddress()==null)
    {
     System.out.println("Platform Address is null");
    }
    return objRootService.getIIOPAddress();
```

```
}
  /**     Newly Linked peer sends its identity and receives peers list.
   @param strMacId
   @param peerAddress
   @return
   @throws RemoteException
   */
  public Hashtable getListPeerAMS(String strMacId, MainAddress objMainAddress)
throws RemoteException
  {
    if ((strMacId != null) && (objMainAddress != null))
    {
      System.out.println("System is returning in getpeerlist " + strMacId);
      objRootService.addPeer(strMacId,objMainAddress);
      Enumeration enum = objRootService.getPeerResolver().keys();
      while (enum.hasMoreElements())
      {
        System.out.println("Server Side Inside getPeerList Host Server Side returning :
" + (String) enum.nextElement());
      }
      return objRootService.getPeerResolver();
    }
    else
      return null;
  }

  /**
   This method gets the String machineId ,adds the machine to the local machine
   Information and returns the machine Information
   @param strMacId
   @return
   @throws RemoteException
   */
  public ArrayList getMachineList(String strMacId) throws RemoteException
  {
    objRootService.addMachineList(strMacId);
    return objRootService.getMachineList();
  }

  /**
   @param alMachineList
   @throws RemoteException
   */
  public void updateMachineList(ArrayList alMachineList) throws RemoteException
  {
    objRootService.setMachineList(alMachineList);
    System.out.println("MACHINE INFORMATION UPDATED");
  }
}
```

D.3 ClientLookup

This is class is also part of Peer Management concern. This is the Client representative class of the AMS RMI. It calls the AMS Server Methods.

```
package ams.peer;

import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Hashtable;

import ams.HostException;
import ams.RootService;

public class ClientLookup {
        private LookupFactory objLookupFactory = null;

        private ActionStatus objActionStatus = null;

        private String strPlatformName = null;

        private RootService objRootService = null;

        /**
         Constructor for ClientLookup which accepts 3 parameters
         @param objRootService
         @param MacId
         @param address
         */
public ClientLookup(RootService objRootService, String MacId, MainAddress address) {
                objRootService.getPeerResolver().put(MacId, address);
                Enumeration enum = objRootService.getPeerResolver().keys();
                while (enum.hasMoreElements())
                        System.out.println("ClientLookup Constructor : " + (String)
enum.nextElement());

                System.out.println("gdgdgfdgfd");

                this.objRootService = objRootService;
                System.out.println("gdgdgfdgfd");
                objLookupFactory = new LookupFactory(objRootService,
objRootService.getAgentPlatformAddress(), objRootService.getAgentPlatformName());
        }

        /**
         Constructor for ClientLookup which accepts 1 parameter
         @param objRootService
         @param MacId
```

```
        @param address
        */
        public ClientLookup(RootService objRootService) {
                this.objRootService = objRootService;
LookupFactory(rs.getPeerResolver(),this.rs.getAgentPlatformAddress(),
this.rs.getAgentPlatformName());
                objLookupFactory = new LookupFactory(objRootService,
this.objRootService.getAgentPlatformAddress(),
this.objRootService.getAgentPlatformName());
        }

        /**
        This method returns the LookupFactory Object
        @return
        */
        public LookupFactory getLookupFactory() {
                return objLookupFactory;
        }

        /**
        This method pings the peer machine and returns the action status based on
        the probe result.
        @param strPeerMacId
        @param strPeerComponent
        @return
        @throws HostException
        */
        public ActionStatus pingPeer(String strPeerMacId, String strPeerComponent)
throws HostException {
                try {
                        System.out.println("I am in the Client Lookup");
                        if (objLookupFactory.getPeerAMS(strPeerMacId) == null) {
                                System.out.println("Peer ams is not null");
                        }
                        objActionStatus =
objLookupFactory.getPeerAMS(strPeerMacId).pingResponse(strPeerComponent);
                        if (objActionStatus == null)
                                return (objActionStatus);
                } catch (Exception e) {

                        System.out.println("Exception Raised in pingPeer : " +
e.toString());
                        this.bindPeerAMS(strPeerMacId);
        }
                return (null);
        }

        /**
        This method pings the machine name which is specified in the arguments
        @param strPeerMacId
```

```
        */
        public void pingPeerAuth(String strPeerMacId) {
                try {
                        objActionStatus =
objLookupFactory.getPeerAMS(strPeerMacId).pingResponse(strPeerMacId);
                } catch (Exception e) {
                        System.out.println("Exception Raised in pingPeerAuth : " +
e.toString());
                }
        }

        /**
         This methos accepts a String machineId of the peer machine and Updates the
         peer machine registry information.
         @param strPeerMacId
         */
        public void getPeerList(String strPeerMacId) {
                Hashtable hashPeerResolver = new Hashtable();
                String strPeerMacIds = new String();

                try {
                        hashPeerResolver =
objLookupFactory.getPeerAMS(strPeerMacId).getListPeerAMS(objRootService.getAgen
tPlatformAddress(), new MainAddress());

                        Enumeration enum = hashPeerResolver.keys();

                        while (enum.hasMoreElements()) {
                                MainAddress ma = null;
                                ma = (MainAddress)
hashPeerResolver.get(enum.nextElement());
                                System.out.println("Client Looku up ===+++ " +
ma.getHostAddress() + "  :  " + ma.getHostPort());
                        }
                } catch (Exception e) {
                        System.out.println("Exception Raised in getPeerList : " +
e.toString());
                }
                System.out.println("Size of HashTable inside getPeerList : " +
hashPeerResolver.size());
                this.objRootService.setPeerResolver(hashPeerResolver);

                Enumeration ePeerResolver =
this.objRootService.getPeerResolver().keys();
                System.out.println("Hello World inside Client lookup");
                while (ePeerResolver.hasMoreElements()) {
                        strPeerMacIds = (String) ePeerResolver.nextElement();
                        if
((strPeerMacIds.equals(this.objRootService.getAgentPlatformAddress())) ||
(strPeerMacIds.equals(strPeerMacId))) {
```

```
                        System.out.println("Already Done");
                    } else {

                        try {
System.out.println("Before Calling for gettingList and adding : " + strPeerMacIds);
                            objActionStatus =
objLookupFactory.getPeerAMS(strPeerMacIds).addPeerAMS(this.objRootService.getAg
entPlatformAddress(), new MainAddress());
                            objActionStatus.setActionStatus("DONE");
                        } catch (Exception e) {
                            System.out.println("Exception Raised in getPeerList
and Add to remote: " + e.toString());
                        }
                    }.
                }
            }

            /**
             This method accepts the String machine Id and removes the machine from the
              local machine
              @param strPeerMacId
             */
            public void removePeer(String strPeerMacId) {
                try {
                    System.out.println("Calling for removing the PeerAMS : " +
strPeerMacId);
                    objActionStatus =
objLookupFactory.getPeerAMS(strPeerMacId).removePeerAMS(strPeerMacId);
                } catch (Exception e) {
                    System.out.println("Exception Raised in removePeer : " +
e.toString());
                }
            }

            /**
              This method accepts a string machine Id and gets the Agent Platform ACC
Address
              from that machine.
              @param strPeerMacId
             */

            //@TODO New change mobility Critical
            public void getACCServerAddress(String strPeerMacId) {
                try {
                    System.out.println("Calling for getting AP Address : " +
strPeerMacId);

            objRootService.setIIOPAddress(objLookupFactory.getPeerAMS(strPeerMacId).g
etACCPlatformAddress());
                    String[] strAddress = objRootService.getIIOPAddress();
```

```
                    System.out.println("Getting agent Platform Address : " +
strAddress[0]);
                } catch (Exception e) {
                    e.printStackTrace();
                }
        }

        /**
         This method accepts a string machine Id and gets the Agent Platform Name
         from that machine.
         @param strPeerMacId
         */
        public void getPlatformName(String strPeerMacId) {
                try {
                    System.out.println("Calling for getting APName : " +
strPeerMacId);

        objRootService.setAgentPlatformName(objLookupFactory.getPeerAMS(strPeerM
acId).getPlatformName());
                    System.out.println("Getting agent Platform Name : " +
objRootService.getAgentPlatformName());
                } catch (Exception e) {
                    System.out.println("Exception Raised in getPlatformName : " +
e.toString());
                }
        }

        /**
         This method accepts the two String arguments and gets the machineList
         from the peer machine .
         @param strPeerMacId
         @param strMacId
         @return
         */
        public ArrayList getMachineList(String strPeerMacId, String strMacId) {
                ArrayList alMachineList = null;
                try {
                    alMachineList =
objLookupFactory.getPeerAMS(strPeerMacId).getMachineList(strMacId);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return alMachineList;
        }

        public void bindPeerAMS(String strMacId) throws HostException {

                Hashtable hashPeerResolvers = null;
                System.out.println("----------I m in the method to handle the error-----
---------" + strMacId);
```

```
                hashPeerResolvers = objRootService.getPeerResolver();
                hashPeerResolvers.remove(strMacId);
                objRootService.getLookupFactory().getPeerCache().remove(strMacId);
                System.out.println("Removed entry from local machine");
                Enumeration enum = hashPeerResolvers.keys();
                while (enum.hasMoreElements()) {
                        try {

        objLookupFactory.getPeerAMS(enum.nextElement().toString()).removeMachine(
strMacId);

        objRootService.getLookupFactory().getPeerCache().remove(strMacId);
                                System.out.println("******////////*********Removed from
machine" + enum.toString());
                        } catch (Exception ex) {
                                ex.printStackTrace();
                                throw new HostException("Host Machine not found");
                        }
                }
        }
}
```

D.4 LookupFactory

This is class is also part of Knowledge Base concern. This is class is also part of Knowledge Base concern. Lookup factory is used in booting. LookupFactory is created by the ClientLookup. Lookup Factory is set and can be obtained from *RootService.* It is required by all the system Agents because they need to communicate with other peer System Agents.

```
package ams.peer;
import java.rmi.Naming;
import java.util.Enumeration;
import java.util.Hashtable;

import mts.peer.PeerMTS;
import vma.AMSGUI.PeerVMA;
import ams.RootService;
import ams.SingletonException;
import ams.Utility;
import df.peer.DFPeer;
public class LookupFactory
{
   private Hashtable hashRegistery; //= new Hashtable();
   private RootService objRootService=null;
   private Hashtable hashPeerCache = new Hashtable();
```

```
static  boolean boolInstance=false;


private String strMahineID = null;

private String strLocalPlatform=null;

/**
 * @param hashRegistery
 */
public LookupFactory()
{
   if(boolInstance)
   throw new SingletonException("Only one Lookup Factory is Allowed");
   else
   {
      boolInstance=true;
   }
}

/**
 *
 * @param hashRegistery
 */
 public LookupFactory(RootService objRootService, String strLocalMacId, String
strPlatformName)
{
   if(boolInstance)
   throw new SingletonException("Only one Lookup Factory is Allowed");
   else
   {
      boolInstance=true;
   }

      this.objRootService = objRootService;
      this.strLocalPlatform = strPlatformName;
      this.strMahineID = strLocalMacId;
      System.out.println("Size in Side LookupFacotry constructor " +
objRootService.getPeerResolver().size());
      this.hashRegistery = objRootService.getPeerResolver();
}

public Hashtable getPeerCache()
{
   return this.hashPeerCache;
}
public void setPlatformName(String strPlatformName)
{
   this.strLocalPlatform = strPlatformName;
```

```
}

public String getPlatformName()
{
   return this.strLocalPlatform;
}

public String getLocalMacId()
{
   return this.strMahineID;
}

/**
 * @param strMachineName
 * @return
 */
public PeerMTS getPeerMTS(String strMachineName)
{
   PeerCache objPeerCache = (PeerCache)hashPeerCache.get(strMachineName);

   if(objPeerCache != null)
   {
      if(objPeerCache.getPeerMTS() == null)
      {
         return( (PeerMTS)lookup(strMachineName, "PeerMTS", objPeerCache) );
      }
      else
      {
         return(objPeerCache.getPeerMTS());
       }
   }
   else
   {
      objPeerCache = new PeerCache();
      return( (PeerMTS)lookup(strMachineName, "PeerMTS", objPeerCache) );
   }
}

/**
 * @param strMachineName
 * @return
 */
public PeerVMA getPeerVMA(String strMachineName)
{
   System.out.println("CHECK 1");
   PeerCache objPeerCache = (PeerCache)hashPeerCache.get(strMachineName);

   if(objPeerCache != null)
   {
      System.out.println("CHECK 2");
```

```
        if(objPeerCache.getPeerVMA()==null)
        {
          return( (PeerVMA)lookup(strMachineName,"PeerVMA", objPeerCache) );
        }
        else
        System.out.println("CHECK 3");
          return(objPeerCache.getPeerVMA());
      }
      else
      {
        System.out.println("CHECK 4");
        objPeerCache = new PeerCache();
        return( (PeerVMA)lookup(strMachineName, "PeerVMA", objPeerCache) );
      }
  }

  /**
   * @param strMachineName
   * @return
   */
  public PeerAMS getPeerAMS(String strMachineName)
  {
    PeerCache objPeerCache = new PeerCache();
    objPeerCache = (PeerCache)hashPeerCache.get(strMachineName);

    if(objPeerCache != null)
    {
       if(objPeerCache.getPeerAMS()==null)
        {
         return( (PeerAMS)lookup(strMachineName, "PeerAMS", objPeerCache) );
        }
       else
        {
         return(objPeerCache.getPeerAMS());
        }
    }
    else
    {
       objPeerCache = new PeerCache();
       try
       {
         return( (PeerAMS)lookup(strMachineName, "PeerAMS", objPeerCache) );
       }
       catch(Exception e)
       {
        System.out.println("Exception in returning PeerAMS Lookup "+e.toString());
        return null;
       }
    }
  }
```

```
/**
 *
 * @param strMachineName
 * @return
 */

public DFPeer getPeerDF(String strMachineName)
{
 PeerCache objPeerCache = new PeerCache();
 objPeerCache = (PeerCache)hashPeerCache.get(strMachineName);
 if(objPeerCache != null)
 {
  if(objPeerCache.getPeerDF()==null)
  {
   return( (DFPeer)lookup(strMachineName, "PeerDF", objPeerCache) );
  }
  else
  {
   return(objPeerCache.getPeerDF());
  }
 }
 else
 {
  objPeerCache = new PeerCache();
  try
  {
   return( (DFPeer)lookup(strMachineName, "PeerDF", objPeerCache) );
  }
  catch(Exception e)
  {
   System.out.println("Exception in returning PeerDF Lookup "+e.toString());
   return null;
  }
 }
}
 /**
  * @param strMachineName
  * @param strServiceName
  * @param objPeerCache
  * @return
  */
 private Object lookup(String strMachineName, String strServiceName, PeerCache
objPeerCache)
 {
   try
   {
     System.out.println("CHECK 1");
     System.out.println("Macvcvdfdfsdfdsfdsfdsf : " +strMachineName);
```

```
        MainAddress objMainAddress =
(MainAddress)this.objRootService.getPeerResolver().get(strMachineName);
        System.out.println("CHECK 2");
        System.out.println("Size in Side LookupFacotry  Hash " +
this.objRootService.getPeerResolver().size());
        System.out.println("CHECK 3");
        System.out.println("Size in Side LookupFacotry  Root " +
this.objRootService.getPeerResolver().size());
        System.out.println("CHECK 4");
        System.out.println("Lookup");
        if(strServiceName.equals("PeerMTS"))
        {

          System.out.println("CHECK 5");
objMainAddress.getHostAddress() + ":" + objMainAddress.getHostPort() + "/" +
strServiceName);
          //PeerMTS peerMTSObject = (PeerMTS)Naming.lookup("rmi://" +
objMainAddress.getHostAddress() + ":" + objMainAddress.getHostPort() + "/" +
strServiceName);
          System.out.println("rmi://" + objMainAddress.getHostAddress() + ":" +
objMainAddress.getHostPort() + "/" + strServiceName);
          System.out.println("rmi://10.10.2.193:1099"+"/"+ strServiceName);
        //   PeerMTS peerMTSObject =
(PeerMTS)Naming.lookup("rmi://10.10.2.193:1099"+"/"+ strServiceName);
          if (peerMTSObject==null)
            System.out.println("Lookup 2");
          else
            System.out.println("Lookup 2 Not Null");
          objPeerCache.setPeerMTS(peerMTSObject);
          hashPeerCache.put(strMachineName, objPeerCache);
          return(peerMTSObject);
        }

        else if(strServiceName.equals("PeerAMS"))
        {

         Enumeration ee = this.objRootService.getPeerResolver().keys();
         System.out.println("Size in Side LookupFacotry  Hash " +
this.objRootService.getPeerResolver().size());
         System.out.println("Size in Side LookupFacotry  Root " +
this.objRootService.getPeerResolver().size());
        while (ee.hasMoreElements())
        {
          MainAddress objMainAdd =null;
          System.out.println("System HAsh SIze :
"+this.objRootService.getPeerResolver().size());
          System.out.println("System Root HAsh SIze :
"+this.objRootService.getPeerResolver().size());
          objMainAdd = (MainAddress)
this.objRootService.getPeerResolver().get(ee.nextElement());
```

```
            System.out.println("Size in Side LookupFacotry  Hash " +
this.objRootService.getPeerResolver().size());
            System.out.println("Size in Side LookupFacotry Root " +
this.objRootService.getPeerResolver().size());
            System.out.println("Lookup FActory ++++===+++ "+
objMainAdd.getHostAddress()+ "  :  " + objMainAdd.getHostPort());
        }
            PeerAMS peerAMSObject = (PeerAMS)Naming.lookup("rmi://" +
objMainAddress.getHostAddress() + ":" + objMainAddress.getHostPort() + "/" +
strServiceName);
            if (peerAMSObject==null)
                System.out.println("Lookup 2");
            else   System.out.println("Lookup 2 Not Null");
            objPeerCache.setPeerAMS(peerAMSObject);
            hashPeerCache.put(strMachineName, objPeerCache);
            return(peerAMSObject);
        }
        else if(strServiceName.equals("PeerVMA"))
        {

            System.out.println("CHECK 6");
            System.out.println("CHECK 6");
            PeerVMA peerVMAObject = (PeerVMA)Naming.lookup("rmi://" +
objMainAddress.getHostAddress() + ":" + objMainAddress.getHostPort() + "/" +
strServiceName);

            if(peerVMAObject!=null)
            System.out.println("NOT NULL"+peerVMAObject);
            System.out.println("CHECK 7");
            objPeerCache.setPeerVMA(peerVMAObject);
            System.out.println("CHECK 8");
            hashPeerCache.put(strMachineName, objPeerCache);
            System.out.println("CHECK 9");
            return(peerVMAObject);
        }

        else if(strServiceName.equals("PeerDF"))
        {

            System.out.println("CHECK 6");
            System.out.println("CHECK 6");
            DFPeer peerDFObject = (DFPeer)Naming.lookup("rmi://" +
objMainAddress.getHostAddress() + ":" + objMainAddress.getHostPort() + "/" +
strServiceName);

            if(peerDFObject!=null)
            System.out.println("NOT NULL"+peerDFObject);
            System.out.println("CHECK 7");
            objPeerCache.setPeerDF(peerDFObject);
            System.out.println("CHECK 8");
```

```
            hashPeerCache.put(strMachineName, objPeerCache);
            System.out.println("CHECK 9");
            return(peerDFObject);
          }
        }
      catch(Exception e)
        {
          System.out.println("Exception in LookupFactory in AMS :" + e);
          System.out.println("have to bind again here");

        }
        return(null);
    }

    public void removePeer(String macId)
    {
      if (hashPeerCache.contains(macId))
        hashPeerCache.remove(macId);
    }

    public void KillVMA()
    {
    try{
      System.out.println("About to remove the peer VMA from hash table");
      PeerCache peerCache=null;
      Utility utility =new Utility();
      String macID =
utility.setMacID(java.net.InetAddress.getLocalHost().getHostAddress());
      peerCache=(PeerCache)hashPeerCache.get(macID);
      peerCache.removePeerVMA();
      System.out.println("Removed the peer VMA from hash table");
      }
    catch(Exception e)
    {
      e.printStackTrace();
    }

    }
}
```

D.5 RootService

This class is also part of Knowledge Base concern. It is created at the time Agent Platform is initiated. RootService is set, in which the reference of the shared registry is passed on to the root service instance. It sets the agent platform description. RootService sets the agent platform name, which is obtained by the system and is the name of the system.

```
package ams;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Vector;

import mobility.MigrationManager;
import mobility.SageClassLoader;
import mts.MTSManager;
import mts.reception.CommunicationModule;
import mts.transmission.SendCommand;
import ams.peer.ActionStatus;
import ams.peer.LookupFactory;
import ams.peer.MainAddress;
//@TODO new change in mobility

public class RootService {
        /**     String Array used to store IIOP Address     */
        private String[] strIIOPAddress;
        /**     String used to store Agent Platform's name  */
        private String strAgentPlatformName;

        /**     String used to store Agent Platform's Description   */
        private String strAgentPlatformDescription;

        /**     String used to store Agent Platform's Address (Unique Machine ID) */
        private String strAgentPlatformAddress;

        /**     Value object for peerResolvers      */
        private MainAddress objPeerAddress = new MainAddress();

        /**     Vector used to store Agent Id's of the Main Service of the Agent
Platform for example Agent Management System , Visual Management Agent.     */
        private Vector vecAgentIDs;

        /**     ActionStatus for the return values  */
        private ActionStatus objActionStatus = new ActionStatus();

        /**     This is used to store a reference of MTS Class.     */
        private MTSManager objMTSManager;

        /**     Hash table which stores the machine id of itself and its peers.     */
        private Hashtable hashPeerResolvers = new Hashtable();

        /**     Array List used to store machine information and keep in consistent*/
        private ArrayList alMachineList = new ArrayList();

        /**     Hash table which is shared between MTS and AMS , it stores the
reference of the agents against the agent names     */
        private Hashtable hashMTSAMSreg = null;
        /**     Used to lookup the peer components on peer machines.    */
```

```
        private LookupFactory objLookupFactory = null;

////////////////////////////////////////////<   Mobility   >////////////////////////////////////////////////////

        private MigrationManager MigrationManagerObj;
        private SageClassLoader objClassLoader;

        /**      Constructor for RootService class    */
        public RootService(Hashtable hashMTSAMSreg) {
                vecAgentIDs = new Vector(0);
                this.setAgentPlatformAddress();
                this.hashMTSAMSreg = hashMTSAMSreg;
                this.addPeer(this.getAgentPlatformAddress(), this.objPeerAddress);
        }

/**      This method returns an instance of the MainAddress
         @return
         */
        public MainAddress getMainAddress() {
                return this.objPeerAddress;
        }

        /**      This method accept the LookUpFactory object and sets it to the Local
LookupFactory attribute
         @param objLookupFactory
         */
        public void setLookupFactory(LookupFactory objLookupFactory) {
                this.objLookupFactory = objLookupFactory;
        }

        /**      This method returns an instance of LookUpFactory
         @return
         */
        public LookupFactory getLookupFactory() {
                return this.objLookupFactory;
        }

        /**      This Method is used to add an AgentId of the Service to the List.
         @param AgentId
         */

        public void addAgentID(AgentId aID) {
                this.vecAgentIDs.add(aID);
        }

        /**      This method accepts an Agent Name and returns the AgentId of that
particular
          Agent.
          @param String strName
        * @return AgentId
```

```
 */

public AgentId getAgentID(String strName) {

        AgentId objAgentId = null;
        for (int i = 0; i < vecAgentIDs.size(); i++) {
                objAgentId = (AgentId) vecAgentIDs.get(i);
                if (objAgentId.getAgentName().equals(strName))
                        return objAgentId;
        }

        return null;
}

/**
 This method accepts the Agent Platform Name and sets it.
 @param String
 */
public void setAgentPlatformName(String strAgentPlatformName) {
        this.strAgentPlatformName = strAgentPlatformName;
}

/**
 * This method returns an instance of the Shared registry Information
 * @return
 */
public Hashtable getSharedRegistry() {

        return this.hashMTSAMSreg;

}

/**
 This method is used to set the Platform Address when a new machine
 joins the platform.
 */
private void setAgentPlatformAddress() {
        String sc = null;
        StringBuffer sb;
        try {

                sc = java.net.InetAddress.getLocalHost().getHostAddress();

                char c[] = sc.toCharArray();
                int point1 = 0;
                int point2 = 0;
                int point3 = 0;
                int count = 0;
                for (int i = 0; i < c.length; i++) {
                        if (c[i] == '.') {
```

```
                                count++;
                                if (count == 1)
                                        point1 = i;
                                if (count == 2)
                                        point2 = i;
                                if (count == 3) {
                                        point3 = i;
                                        break;
                                }
                        }
                }

                sb = new StringBuffer(sc);

                sb.deleteCharAt(point3);
                sb.deleteCharAt(point2);
                sb.deleteCharAt(point1);
                sb = sb.reverse();
                sc = sb.toString();

                System.out.println(sc);

        } catch (Exception e) {
                System.out.println(e.toString());
        }
        this.strAgentPlatformAddress = sc;
}

/**
 This method is used to get the Agent Platform Name.
 @return String
 */

public String getAgentPlatformName() {

        return strAgentPlatformName;
}

/**
 This method is used to get the Agent Platform Address.
 @return String
 */
public String getAgentPlatformAddress() {
        return strAgentPlatformAddress;
}

/**
 This method accepts the Agent Platforms Description and sets it.
 @param agentPlatformDescription
 */
```

```
        public void setAgentPlatformDescription(String strAgentPlatformDescription) {
                this.strAgentPlatformDescription = strAgentPlatformDescription;
        }

        /**
         This method is used to get the Agent Platform's Description
         @return String
         */
        public String getAgentPlatformDescription() {
                return strAgentPlatformDescription;
        }

        /**
         This method accepts a reference of MTS and sets it.
         @param MTS objMTSManager
         */

        public void setMessageTransportService(MTSManager objMTSManager) {
                if (this.objMTSManager == null)
                        this.objMTSManager = objMTSManager;
        } //end of method setMTS

        public MTSManager getMessageTransportService() {
                return this.objMTSManager;
        }

        /**
         This method returns the name of Agent Management System.
         @return String
         */
        public synchronized String getAMS() {
                return returnName("AMS" + ":" + this.getAgentPlatformAddress() + "@"
+ this.getAgentPlatformName());

        } //end of method getAMS()

        /**
         This method returns the name of Visual Management Agent.
         @return String
         */
        public synchronized String getVMA() {
                return returnName("VMA" + ":" + this.getAgentPlatformAddress() + "@"
+ this.getAgentPlatformName());
        } //end of method getVMA()

        /**
         This method returns the name of the Directory Facilitator.
         @return String
         */
        public String getDF() {
```

```
            return returnName("DF" + ":" + this.getAgentPlatformAddress() + "@" +
this.getAgentPlatformName());
        } //end of method getDF()

        /**
         This method accepts the fully qualified name of the Agent and
         returns the name of only the agent.
         @param strAgentName
         @return
         */
        private String returnName(String strAgentName) {
            AgentId objAgentID = null;

            for (int i = 0; i < vecAgentIDs.size(); i++) {
                objAgentID = (AgentId) vecAgentIDs.get(i);

                if (objAgentID.getAgentName().equals(strAgentName))
                    return objAgentID.getAgentName();

            } //end of for loop
//in the future code..... the arguments must be AgentId object instead of the string name
            return null;
        } //end of method returnName()

        /**
         This method accepts an Agent Name and returns a reference of Communication
Module
         associated with that Agent Name.
         @param String stragentName
         @return CommunicationModule
         */
        public CommunicationModule createCommunicationModule(String
strAgentName, ServiceAgent objAgent) {
            return new CommunicationModule(objMTSManager, strAgentName,
objAgent);
        } //end of method createCommunicationModule()

        /**
         This method returns a reference of Send Command
         @return
         */
        public SendCommand createSendCommand() {
            return new SendCommand(objMTSManager);

        } //end of method createSendCommand()

        /**
         Add peer entry in hashPeerResolvers
         */
```

```
    public ActionStatus addPeer(String strPeerMacID, MainAddress
objPeerMainAddress) {
        if ((strPeerMacID != null) && (objPeerMainAddress != null)) {
            hashPeerResolvers.put(strPeerMacID, objPeerMainAddress);
            objActionStatus.setActionStatus("DONE");

        } else
            objActionStatus.setActionStatus("NOTDONE");

        return objActionStatus;
    }

    /**
     get peer entry from hashPeerResolvers
     */
    public MainAddress getPeer(String strPeerMacID) {
        if (strPeerMacID != null) {
            return (MainAddress) hashPeerResolvers.get(strPeerMacID);

        } else
            return null;
    }

    /**
     Remove Peer from Resolver list.
     @param peerMacID
     @return
     */
    public ActionStatus removePeer(String strPeerMacID) {
        if (strPeerMacID != null) {
            if (hashPeerResolvers.contains(strPeerMacID)) {
                hashPeerResolvers.remove(strPeerMacID);
                objActionStatus.setActionStatus("REMOVED");
                this.objLookupFactory.removePeer(strPeerMacID);
            } else
                objActionStatus.setActionStatus("DOESNOTEXIST");
        } else
            objActionStatus.setActionStatus("NULLNAME");
        return objActionStatus;
    }

    /**
     get complete clone of PeerResolver
     */
    public Hashtable getPeerResolver() {
        /**
        Hashtable peersclone = new Hashtable();
        peersclone = (Hashtable) hashPeerResolvers.clone();
        return peersclone;
        */
```

```
        return hashPeerResolvers;
    }
    public void setPeerResolver(Hashtable hashPeerResolvers) {
        System.out.println("RootService " + hashPeerResolvers.size());
        this.hashPeerResolvers = hashPeerResolvers;
        System.out.println("RootService Peer" + this.hashPeerResolvers.size());
        //this.hashPeerResolvers = (Hashtable) ht.clone();
    }

    /**
     This method accepts a machine Id and adds it to the ArrayList
     @param macId
     */
    public void addMachineList(String macId) {
        alMachineList.add(macId);
    }

    /**
     This method is used to get the machine list of the platform
     @return
     */
    public ArrayList getMachineList() {
        return this.alMachineList;
    }

    /**
     This method accepts an attribute machineList of type ArrayList and
     sets it to the local machineList.
     @param alMachineList
     */
    public void setMachineList(ArrayList alMachineList) {
        this.alMachineList = alMachineList;
    }

    /**
     Method for Setting the ACC Server Address
     */
    public void setIIOPAddress(String[] strIIOPAddress) {
        this.strIIOPAddress = strIIOPAddress;
    }

    /**
     Method for getting the ACC Server Address
     @param strIIOPAddress
     */
    public String[] getIIOPAddress() {
        return strIIOPAddress;
    }

////////////////////////////////////////< Mobility ////////////////////////////////////////////////
```

```
    public void setMigrationManager(MigrationManager migrationManagerObj) {
        this.MigrationManagerObj = migrationManagerObj;
    }
    public MigrationManager getMigrationManager() {
        return this.MigrationManagerObj;
    }

    public void setClassLoader(SageClassLoader objSageClassLoader) {
        this.objClassLoader = objSageClassLoader;
    }
    public SageClassLoader getClassLoader() {
        return this.objClassLoader;
    }
    ////////////////////////////////////////////////////////////////////////////////////////////////////
```

} //end of class RootService

D.6 AgentDirectoryService

This is class is also part of Agent Management concern. This is the main Class of Agent Management System, It contains the methods to supervise the agent Platform. More over this is also responsible of taking actions to control the Life Cycle of the agents.

```
package ams;

import java.rmi.RemoteException;
import java.util.Enumeration;
import java.util.Hashtable;

import vma.AMSGUI.VMAGUI;
import acl.ACLMessage;
import acl.ACLPerformatives;
import acl.CFAggregate;
import acl.CFContent;
import acl.CFPredicate;
import acl.CFPrimitive;
import acl.ontology.BasicOntology;
import acl.ontology.management.ManagementOntology;
import acl.sl.codec.SLTokenizer;
import ams.lifecycle.MessageTransceiver;
import ams.peer.ActionStatus;
import ams.peer.ClientLookup;
import ams.peer.PeerAMS;
import ams.peer.PeerServerImpl;
import ams.probe.dynamic.DynamicPolicy;

import df.DirectoryFacilitator;
```

```
public class AgentDirectoryService extends ServiceAgent implements Services {

        AgentLifeCycleManager objManageLifeCycle;
        private String strName;
        private String strDesc = null;
        private String strType = null;
        static boolean boolInstance = false;

        static private String strToken;

        ////////////// CONTENTS OF ACL MESSAGE/////////////////
        String strFileName;
        String strAgentName;
        String strTransportAddresses[];
        String strLocator[];
        String strDescription;
        byte byteState = AgentStates.ACTIVE;

        /**
* This will contain the list of Agent Id's of the agents currently registered on the platform
         */

        PeerServerImpl AMSpeerSever;
        private String strOwnerShip; //Owner of the agent platform
        private String strStateAp; //State of the platform
        private RootService objRootService;

        /**
         * a shared registry of AMS and MTS
         * AMS will keep AID and Agents References
         * in the AgentRegistry class and put it in this hashtable shared among AMS and
MTS
         */
        private AMSMachineInformation objMachineInformation;
        private Hashtable hashAgentRef;
        private ClientLookup objClientLookup;
        /**
         * Constructor of AgentDirectoryService Class , it accepts an AgentId and
         * sets it . It also activates the Service.
         * @param aID
         * @throws SingletonException
         */

        /**
         *This method is used to activate the Service. Its implemetation depends
         *entirely upon the nature of the service and how it is going to be used.
         */
        public void activate() {
                //overloaded method
```

```
}

/**
 *This method accepts an Agent Id of an agent and registers it with AMS
 *It returns a String indicating failure or success
 * @param aID
 * @return
 */
private String registerAgent(AgentId aID) {
        /**the assumption is that agent is already created on the platform
        //and now it wants to register with the platform
        // But we will also need the reference of the Agent.

        /**for next phase Resolvers behavior */
        /** Where an agent physically will exist on another *****/
        /** HAP, but want to register with this platform, to make*/
        /** this platform as Resolver*/
        return null;
}

/**
 This is a utility method which converts an arraylist to a string array
 */

public void run() {
        try {
                /*****************************************************/
                while (true) {
                        // Waiting for Payload, upto that time in wait state.
                        ACLMessage objACLMessage = this.blockingReceive();
                        if (objACLMessage != null) {
                        System.out.println("Agent directory Service
**********************************************************");
objMessageTransceiver.AnalyzeMessage(objACLMessage);
                        }
                }
        } catch (Exception e) {
                e.printStackTrace();
        }
}

/*****************************************************/
//@TODO NEw Change mobility not understandable empty method
public void RebindPeerAMS() {

}

/**
 This method accepts Peer component name , Peer Machine name , ClientLookup
Object. It then instantiates a new
```

```
        Object of CheckPeer class , which is used to check the peer machine.
        @param strPeerComponent
        @param strMessageTransceiver
        @param objClientLookup
        */
       public void ProbeMachine(String strPeerComponent, String
strMessageTransceiver, ClientLookup objClientLookup) {
              CheckPeer checkPeer = new CheckPeer();
              checkPeer.CheckPeerMachine(strPeerComponent, strMessageTransceiver,
objClientLookup);
              //This method will be called in case of a fault in peer machine
       }

       /**
        This method accepts an Agent ID and de registers that agent from Agent
DirectoryService
        @param aID
        @return
        */
       private String deRegisterAgent(AgentId objAgentId) {
              //Check for the reference of the Agent ID against the reference
              //get the reference and delete the AgentId and its reference from the
              //list and also stop the Agent
              // checking if the AgentID is valid
              if (hashAgentRef.get(objAgentId.getAgentName()) == null) {
                     return "No Agent found with the corresponding AgentID";
              }
              hashAgentRef.remove(objAgentId.getAgentName());
              return "Agent deregistered Successfully";
       }

       /**
        This method accepts and Agent ID and modifies that agents description
        @param aID
        @return string
        */
       private String modifyAgentDescription(AgentId objAgentId) {

              // First of all get the AgentID object from hash table
              AgentRegistry objAgentReg = (AgentRegistry)
hashAgentRef.get(objAgentId.getAgentName());
              if (objAgentReg == null)
                     return " No Match found for AgentID";
              else {
                     objAgentReg.setAgentId(objAgentId);
                     hashAgentRef.put(objAgentId.getAgentName(), objAgentReg);
              }
              return "Description Modified";
       }
```

```
/**
 This method accepts the Agent Id and deletes the corresponding agent

@param aID
@returns string
*/
private String deleteAgent(AgentId objAgentId) {
        //Check for the reference of the Agent ID against the supplied reference
        //get the reference and delete the AgentId and its reference from the
        //list and also stop the Agent
        String strAgentName = null;
        strAgentName = objAgentId.getAgentName();
        ServiceAgent objServiceAgent = null;

        if (strAgentName == null) {
                return "Agent Not Found in the list";
        }

        if (hashAgentRef.get(objAgentId.getAgentName()) != null) {

objManageLifeCycle.terminateAgent(objAgentId.getAgentName());
                hashAgentRef.remove(objAgentId.getAgentName());
                return "Agent Successfully deleted";
        } else
                return "Agent Could not be deleted";
}

/**
 This method will accept the Agent id and will search the agents description

 @param aID
@return
*/
public AgentId searchAgentDescription(String strAgentName) throws
RemoteException {

        Utility utility = new Utility();
        String strMachineId = utility.getMachineName(strAgentName);

        Enumeration enum = objRootService.getPeerResolver().keys();
        ActionStatus as;
        AgentId objAgentId = null;
        PeerAMS peerams = null;
        if (strMachineId.equals(objRootService.getAgentPlatformAddress())) {
                AgentRegistry agentRegObj = (AgentRegistry)
hashAgentRef.get(strAgentName);
                if (agentRegObj.getAgentId() == null)
                        System.out.println("null");
                else {
```

```
                    }
                } else {
                    enum = objRootService.getPeerResolver().keys();

                    boolean boolKeys =
objRootService.getPeerResolver().containsKey(strMachineId);
                if (objRootService.getPeerResolver().containsKey(strMachineId)) {
        objAgentId = (AgentId)
objRootService.getLookupFactory().getPeerAMS(strMachineId).searchAgentDesc(strAg
entName);
                    } else {
                    }
                }
                return objAgentId;
        }

        /**
         This method will return the Agent Id of the Agent Management System

         @returns Agent ID
         */

        public AgentId getAgentId() {
                return this.objAgentId;
        }

        /**
         This method will accept the Agents Name and will return the Agent ID.
         If no agent is found this method will return null.
         @param strAgentName
         @return
         */
        private AgentId getAgentId(String strAgentName) {
                AgentRegistry objAgentReg = (AgentRegistry)
hashAgentRef.get(strAgentName);

                if (objAgentReg != null) {
                    return objAgentReg.getAgentId();
                }
                return null;
        }

        /**
         This method will accept the Agent Id and will return its description
         @param aID
         @return
         */
        public AgentId getDescription(String strAgentName) {
                // check if local or remote ...
```

```
            AgentRegistry objAgentReg = (AgentRegistry)
hashAgentRef.get(strAgentName);
            if (objAgentReg != null)
                  return objAgentReg.getAgentId();

            return null;
      }

      /**
       This method will accept the Agent Id and a description of that agent.
       It will store/replace the description in the Agent Id. If no Agent is found
       with the given Agent Id , it will return an error ( STILL TO BE DECIDED )
      @param strDescription
      @param aID
      @return
      */

      private boolean setDescription(String strDescription, AgentId objAgentId) {
            AgentRegistry objAgentReg = (AgentRegistry)
hashAgentRef.get(objAgentId.getAgentName());

            if (objAgentReg == null) {
                  return false;
            }
            objAgentReg.getAgentId().setDescription(strDescription);
            hashAgentRef.put(objAgentReg.getAgentId().getAgentName(),
objAgentReg);
            return true;
      }

      /**
      This method returns the name of the Service
       @return
      */
      public String getServiceName() {
            return this.strName;
      }

      /**
       This method sets the name of the service with the parameter passed by the
       name
       @param name
       */
      public void setServiceName(String strName) {
            this.strName = strName;
      }

      /**
       This method is used to activate the service and register with the platform
       This method also creates the VMA and resgisters it as well.
```

```
        */
        private void activate(AgentId objAgentId) {

                try {
                        objManageLifeCycle = new
AgentLifeCycleManager(this.hashAgentRef);
                        AgentRegistry objAgentReg = (AgentRegistry)
hashAgentRef.get(this.objAgentId.getAgentName());
                        objAgentReg.setAgentId(objAgentId);
                        objAgentReg.setOwner("Owner");
                        hashAgentRef.put(this.objAgentId.getAgentName(),
objAgentReg);

                        AgentId vmaID = new AgentId("VMA", strTransportAddresses,
strLocator, "description of VMA", AgentStates.ACTIVE);
                        System.out.println("in activate method");
                        vmaID.setAgentName(vmaID.getAgentName() + ":" +
objRootService.getAgentPlatformAddress() + "@" +
objRootService.getAgentPlatformName());

                        objRootService.addAgentID(vmaID);
                        objMachineInformation.addAgentActive();
                        objMachineInformation.addAgentCreated();
                        objMachineInformation.addAgentCreated();
                        objMachineInformation.addAgentActive();

                        AgentId dfID = new AgentId("DF", strTransportAddresses,
strLocator, "description of VMA", AgentStates.ACTIVE);
                        dfID.setAgentName(dfID.getAgentName() + ":" +
objRootService.getAgentPlatformAddress() + "@" +
objRootService.getAgentPlatformName());

                        objRootService.addAgentID(dfID);
                        objMachineInformation.addAgentCreated();
                        objMachineInformation.addAgentActive();

                        objAgentReg = (AgentRegistry)
hashAgentRef.get(vmaID.getAgentName());
                        objAgentReg.setOwner("Owner");
                        objAgentReg.setAgentId(vmaID);
                        objAgentReg.setServiceAgent((ServiceAgent) vmaGui);

                        objAgentReg = (AgentRegistry)
hashAgentRef.get(dfID.getAgentName());
                        objAgentReg.setAgentId(dfID);
                        objAgentReg.setOwner("Owner");
                        objAgentReg.setServiceAgent((ServiceAgent) df);

                        System.out.println("In activate methid of VMA AND DF");
ACLMessage objACLMessage = new ACLMessage(ACLPerformatives.INFORM);
```

```
        objACLMessage.setOntology(ManagementOntology.getInstance().getName());
                        objACLMessage.addReceiver(vmaID);
                        objACLMessage.setSender(objAgentId);

                        CFPredicate objRegisteredAgents = new
CFPredicate(ManagementOntology.REGISTEREDAGENTS);
                        CFAggregate registeredAgents = new
CFAggregate(BasicOntology.SEQUENCE);

        registeredAgents.add(CFPrimitive.getCFPrimitiveFor(dfID.getAgentName()));

        registeredAgents.add(CFPrimitive.getCFPrimitiveFor(vmaID.getAgentName()));

        registeredAgents.add(CFPrimitive.getCFPrimitiveFor(objAgentId.getAgentName(
)));
objRegisteredAgents.set(ManagementOntology.AGENT_NAMES, registeredAgents);
                        CFContent content = (CFContent) objRegisteredAgents;
        SLTokenizer slt = new SLTokenizer(ManagementOntology.getInstance());
                        objACLMessage.setContent(slt.encode(content));

                        this.sendMessage(objACLMessage);
                        DynamicPolicy dynamicPolicy = new DynamicPolicy(this);
                        this.start();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        /**
         This method is used to deactivate the service
         */
        public void deActivate() {
        }
                public static String getFirstTok() {
                return strToken;
        }

        /**
         This method accepts Service Description and is used to set it .
         @param String
         */

        public String getServiceDesc() {
                return this.strDesc;
        }

        /**
         This method accepts the description and sets the Service Description.
         @param desc
```

```
        */
        public void setServiceDesc(String strDesc) {
                this.strDesc = strDesc;
        }
        /**
         This method returns the Service Type.
         @return
        */
        public String getServiceType() {
                return this.strType;
        }
        /**
         This method accepts the Service type and sets it .
         @param strType
        */
        public void setServiceType(String strType) {
                this.strType = strType;

        }

        /**
         This method returns the shared Hashtable reference
         @return
        */
        public Hashtable getHashAgentRef() {
                return this.hashAgentRef;
        }

        /**
         This method returns the current instance of RootService
         @return
        */
        public RootService getRootService() {
                return this.objRootService;
        }
        AMSMessageModel amsMessageModel;
}
```

D.7 ServiceAgent

This is class is also part of Agent Management concern. Service Agent is basic class as all agent classes extends from it.

```
package ams;
import mts.reception.CommunicationModule;
import mts.transmission.SendCommand;
import acl.ACLMessage;
```

```
//@TODO New change mobility

public class ServiceAgent extends Thread implements Runnable, Cloneable
{
 public Object clone() throws CloneNotSupportedException
 {
 return super.clone();
 }

 /**  Used to store CommunicationModule reference , which is required to
  recieve Messages across platform.  */

 private CommunicationModule commModule;

 /**  Used to store SendCommand reference , which is used to Send Messages
  across Platform.  */
 private SendCommand sendCommand;

 /**  Used to store the reference of the AgentId , which is unique across the
  platform.  */
 protected AgentId objAgentId;

 public boolean boolWaitState;

 public ServiceAgent()
 {
 }

 /**
 Constructor for ServiceAgent Class , It accepts a AgentId reference and sets it to its own
AgentId. It gets the CommunicationModule and SendCommand reference and sets them
so that the Agent can send and recieve messages.
  @param agentId
 */
 public ServiceAgent( AgentId objAgentId )
 {
   this.objAgentId=objAgentId;

   commModule =
RSFactory.createCommunicationModule(objAgentId.getAgentName() , this );
   sendCommand = RSFactory.createSendCommand();

 }
 /**
  Sends the message to the Message Transport Service
  so that it can be sent towards destination
  @param payload
 */
 public void sendMessage( ACLMessage message )
 {
```

```
        message.setSender(this.objAgentId);
        try
        {
         sendCommand.execute( message );
        }
        catch(Exception e)
        {
           e.printStackTrace();
        }
    }
    /**
     Returns the payload present in the queue of the Agent.
     It returns null if there is no message in the queue
     @return Payload
     */
    protected ACLMessage getMessage()
    {
        return commModule.getMessage();
    }

    public ACLMessage blockingReceive()
    {
        return getMessage();

    }//end of method blockingReceive

    public AgentId getAgentId()
    {
    return this.objAgentId;
    }
    public void Resume(){
    }
    public void run(){
    }
}
```

D.8 KnowledgeDistributionAspect

This aspect is associated with the classes of Peer Management concern, Knowledge Base concern, and Agent Management concern. It captures the joinpoints of RootService AgentDirectoryService and LookupFactory classes from the whole system.

```
package ams;


import java.util.ArrayList;
import java.util.Vector;
```

```
import java.util.Enumeration;
import java.util.Hashtable;
import java.io.Serializable;

aspect KnowledgeDistributionAspect {

/** setActionStatus method*/
pointcut setStatus():call (public void
ActionSatus.setActionStatus(String))&&!within(KnowledgeDistributionAspect);

after(): setStatus() {}

/** getLookupFactory method */
pointcut lookup(): call(public
RootService.getLookupFactory())&&!within(KnowledgeDistributionAspect);

after() : lookup() {}


/** getPeerResolver method */
pointcut PeerR(): call(public
Hashtable.getPeerResolver())&&!within(KnowledgeDistributionAspect);

around () : PeerR() { }

/** getPeerCache method */
pointcut PeerC(): call(public
Hashtable.getPeerCache())&&!within(KnowledgeDistributionAspect);

Hashtable around () : PeerC() { }

/** getRootService method */
pointcut RS(): call(public RootService
AgentDirectoryService.getRootService())&&!within(KnowledgeDistributionAspect);

around () : RS() { }

/** aspect ends here*/

}
```

D.9 KnowledgeConsistencyAspect

This aspect is associated with the classes of Peer Management concern, Knowledge Base concern, and Agent Management concern. It captures the joinpoints of RootService and LookupFactory classes from the whole system.

```
package ams;
import ams.peer.ActionStatus;
import ams.probe.dynamic;
import ams.peer.LookupFactory;

import java.util.ArrayList;
import java.util.Vector;
import java.util.Enumeration;
import java.util.Hashtable;
import java.io.Serializable;

aspect KnowledgeConsistencyAspect {

/** getActionStatus method*/
pointcut getStatus():call (*
ActionSatus.getActionStatus())&&!within(KnowledgeConsistencyAspect);

String around(): getStatus() {}

/** getAgentPlatformName and Address methods */
pointcut getAP(): call(public String
getAgentPlatform*())&&!within(KnowledgeConsistencyAspect);

around() : getAP() {}


/** getPeerAMS method */
pointcut PeerAMS(): call(public PeerAMS
LookupFactory.getPeerAMS(String))&&!within(KnowledgeConsistencyAspect);

around () : PeerAMS() { }

/** aspect ends here*/

}
```

D.10 ExceptionAspect

This aspect is associated with the classes of Peer Management concern, Knowledge Base concern, and Agent Management concern. It captures joinpoints of exception handling from all the 25 classes of the sytem.

```
aspect ExceptionAspect {
        after () throwing (Throwable ex) : call (* *.*(..)) && !within(ExceptionAspect)
        {
                ex.printStackTrace(System.err);
        }

/** aspect ends here*/
}
```