|  |
| --- |
| DSV Logo |
| Security for Collaboration of Mobile Agents |
|  |
|  |
|  |
|  |

KASHIF DAR

Department of Computer and System Sciences (DSV)

This thesis corresponds to 20 weeks of full-time work.

|  |
| --- |
| Mobile Agent, Collaboration, FIPA, MASIF, Authentication, Message Security, PKCS 7, SAGE, Java Socket Programming, Eclipse RCP. |

**ABSTRACT**

Mobile agent is reffered to as a process (software agent) that can migrate from one envirinment to another while keeping its data, and still being able to resume and execute in another environment. Mobile agents reduce network traffic as compared to convential client/server applications, because they can be dispatched from central controller to act locally.

Mobile agent being capable of *social ability*, has to share information with other agents to work efficiently. However there are two main issues of this collaboration: first is to define message format that will be acceptable to a variety of different vendors, and second is the security threats involved during their collaboration e.g., masquerading, information disclosure, information integrity, trustworthiness of information sender and receiver etc.

The solution for the first problem has been provided by Foundation of Intelligent Physical Agents (FIPA) that descibed standard Agent Communiocation Language (ACL) which can be used to exchange messages between agents. Consequently, two agents can communicate with each other through FIPA compiant ACL messages. FIPA also specifies the ontology that is treated as a shared vocabulary among the community of agents and is used to construct ACL messages.

In oreder to provide agents with a secure collaborative environment in which they can comfortably make conversation, first they have to authenticate each other, secondly they should exchange messages without loss of data confidentialty and integrity. To achieve this goal, Public Key Cryptographic Standard 7 (PKCS 7) is followed. PKCS 7 signed and enveloped data provides all these security features.

Thanks to SAGE, the open source agents platform provided by National Institute of Information Technology (Islamabad, Pakistan) whose ACL and Ontology modules were re-used to implemnt FIPA compliant ACL messages and defining ontology for the domain of our interest.

Finally, a simple File Search Application (used to search for a particular file on different remote hosts) is designed and implemented to realize the outcome of our work. This application uses frequent communication between the working agents during their lifetime.

Despite all the effort, there are still improvements that has to be done. The other possible threats that have not been addressed in this work include the proper authorization mechanism to control access to the sensitive information contained by agents, the issue of delegation when agents produce their clones, message flooding by the malicios agents for denial of service attack, and the repudiation attacks.

**ACKNOWLEDGEMENTS**

This dissertation could not have been written without Prof. Dr. Sead Muftic who not only served as my supervisor, but also encouraged and challenged me throughout my academic program. He always guided me in the right direction through his unique way that is easily understandable.

I would also like to express my gratitude to Mr. Awais Shibli and Mr. Abdul Ghafoor, PhD students in the SecLab. They guided me nicely through the dissertation process, discussing valuable ideas about my work, giving their priceless suggestions, and sacrificing their precious time to support me.

I would appreciate to my other collegues in the SecLab doing master thesis, Pasquale Stirparo from Italy, Javier Margado from Spain, Feng Zhang from China, Qing Song from China, He Huang from China. I not only gained some valueable technical thoughts from them, but also experienced a multi-culture environment first time in my life. I really enjoyed working with them.

I will also give a tribute to all of my family members in Pakistan and my maternal uncle and aunt in Italy

who not only supported me, but also encouraged me to complete my studies. I really missed my Pakistani family from the soul of my heart during the one and a half year stay in Sweden. However, my home sickness was significantly reduced during my visit in summer vacations to see my maternal uncle and his family in Italy. I am very gratefull for their hospitality and nice company.

At the end, once again, I would like to thank the whole team of SecLab that gave me such a platform to pursue this innovative activity.

**DEDICATIONS**

***I dedicate this work:***

**To my Maternal Grandfather: M. Fazal Butt.**

**To whom I lost during my stay in Sweden, and I couldn’t even say him *“KHUDA HAFIZ (Goodbye).”* For me, he was more friend than grandfather. May God bless him and rest his soul into the heaven.**

**To my Paternal Grandfather: Saddar-ud-Deen Dar.**

**Who flame the light of knowledge into our family and always preferred to seek knowledge to lessen the inner darkness. May God bless him too and rest his soul into the heaven.**

Table of Contents

[1. INTRODUCTION OF THE THESIS 8](#_Toc194236846)

[1.1 Introduction- Motives and Goals 8](#_Toc194236847)

[1.2 Statement of the problem 9](#_Toc194236848)

[1.2.1 Defining Message formats 9](#_Toc194236849)

[1.2.2 Identify potential threats 9](#_Toc194236850)

[1.2.3 Building Solution 9](#_Toc194236851)

[1.2.4 Implementation 10](#_Toc194236852)

[1.3 The Goals of the Project 10](#_Toc194236853)

[1.4 The Road Map 10](#_Toc194236854)

[1.5 Relevant Standards and Background Material 10](#_Toc194236855)

[1.6 Scope of the Problem Area 12](#_Toc194236856)

[1.7 Schedule 12](#_Toc194236857)

[2. BACKGROUND, RELEVANT STANDARDS AND TECHNOLOGIES 13](#_Toc194236858)

[2.1 Why Study Standards and Technologies 13](#_Toc194236859)

[2.2 About Mobile Agents 13](#_Toc194236860)

[2.2.1 What is Mobile Agent? 13](#_Toc194236861)

[2.2.2 Mobile Agents Team Organization 14](#_Toc194236862)

[2.2.3 Agents Team Models 15](#_Toc194236863)

[2.3 Foundation for Intelligent Physical Agents (FIPA) 16](#_Toc194236864)

[2.3.1 Agent Communication Language (ACL) Structure 16](#_Toc194236865)

[2.3.2 Agent Security Management Specifications (FIPA ASMS) 17](#_Toc194236866)

[2.3.3 FIPA Agent Communication 19](#_Toc194236867)

[2.3.4 FIPA Agent Communication Security 20](#_Toc194236868)

[2.3.5 FIPA Auditing 21](#_Toc194236869)

[2.3.6 FIPA Ontology Service Specification 21](#_Toc194236870)

[2.4 Mobile Agent System Interoperability Facility (MASIF) 21](#_Toc194236871)

[2.5 Public Key Cryptographic Standard 7 (PKCS 7) 22](#_Toc194236872)

[2.6 About Technology 23](#_Toc194236873)

[2.6.1 Eclipse with Rich Client Platform (RCP) Plug-ins 23](#_Toc194236874)

[2.6.2 Scalable, Fault Tolerant Agent Grooming Environment (SAGE) 23](#_Toc194236875)

[3. THREATS AND SECURITY REQUIREMENTS 24](#_Toc194236876)

[3.1 Potential Threats 24](#_Toc194236877)

[3.1.1 Disclosure 24](#_Toc194236878)

[3.1.2 Authentication of Source and Target 24](#_Toc194236879)

[3.1.3 Alteration 24](#_Toc194236880)

[3.1.4 Copy and Replay 24](#_Toc194236881)

[3.1.5 Spoofing and Masquerading 24](#_Toc194236882)

[3.1.6 Denial of Service (DoS) 24](#_Toc194236883)

[3.1.7 Repudiation 25](#_Toc194236884)

[3.2 Security Requirements 25](#_Toc194236885)

[3.2.1 Authentication 25](#_Toc194236886)

[3.2.2 Data Confidentiality 25](#_Toc194236887)

[3.2.3 Message Integrity 25](#_Toc194236888)

[3.2.4 Source Integrity 26](#_Toc194236889)

[4. DESIGN OF THE SECURITY SYSTEM 27](#_Toc194236890)

[4.1 Creating FIPA Compliant ACL Messages 27](#_Toc194236891)

[4.1.1 SAGE ACL Module Engine Classes 27](#_Toc194236892)

[4.1.2 Custom Designed Classes for Authentication Messages 28](#_Toc194236893)

[4.1.3 Example: Creating Simple ACL Message for Authentication Challenge 29](#_Toc194236894)

[4.2 Authentication Protocol 31](#_Toc194236895)

[4.3 PKCS 7 Message Enveloping 33](#_Toc194236896)

[4.4 Data Communication and Code Mobility 35](#_Toc194236897)

[4.5 File Search Application 37](#_Toc194236898)

[4.6 Ontology Design 39](#_Toc194236899)

[5. IMPLEMENTATION 41](#_Toc194236900)

[5.1 Creating ACL Messages 41](#_Toc194236901)

[5.1.1 ACL Message Factory Class (ACLMsgFactory.java) 41](#_Toc194236902)

[5.1.2 Search Result ACL Messages Class (SearchResulACLMessage.java) 41](#_Toc194236903)

[5.2 Implementation of Authentication Protocol and Authentication Class (Authentication.java) 41](#_Toc194236904)

[5.3 PKCS 7 Implementation (PKCS\_7.java) 41](#_Toc194236905)

[5.4 Data Communication and Code Mobility Implementation 42](#_Toc194236906)

[5.5 Agents Implementation in File Search Application 42](#_Toc194236907)

[5.6 Ontology Implementation 42](#_Toc194236908)

[6. Conclusions and Future Work 43](#_Toc194236909)

[6.1 Conclusions 43](#_Toc194236910)

[6.2 Future Work 44](#_Toc194236911)

[6.2.1 Cascading Delegation Problem 44](#_Toc194236912)

[6.2.2 Authorization Control 44](#_Toc194236913)

[6.2.3 Repudiation 45](#_Toc194236914)

[6.2.4 Message Flooding 45](#_Toc194236915)

[7. Appendix A: Creating, Enveloping and Sending ACL Messages 46](#_Toc194236916)

[7.1 ACL message Factory Class (ACLMessageFactory.java) 46](#_Toc194236917)

[7.2 File Search Result ACL Messages (SearchResultACLMessage.java) 46](#_Toc194236918)

[7.3 Authentication.java 48](#_Toc194236919)

[7.4 PKCS\_7.java 49](#_Toc194236920)

[7.5 Server.java 49](#_Toc194236921)

[7.6 SocketAction.java 49](#_Toc194236922)

[7.7 ServerConnectionThread.java 50](#_Toc194236923)

[8. Appendix B: Agents Implementation In File Search Application 51](#_Toc194236924)

[8.1.1 Agent Class Agent.java) 51](#_Toc194236925)

[8.1.2 Master Agent Class (MasterAgent.java) 51](#_Toc194236926)

[8.1.3 Messenger Agent Class (Messenger.java) 51](#_Toc194236927)

[8.1.4 Mobile Agents Team Class (MATeam.java) 51](#_Toc194236928)

[8.1.5 Agent Team Lead Class (TeamLead.java) 52](#_Toc194236929)

[8.1.6 Agent Team Member Class (TeamMember.java) 52](#_Toc194236930)

[9. Appendix C: Ontology Implementation for File Search Application 53](#_Toc194236931)

[9.1 Goal Ontology 53](#_Toc194236932)

[9.2 Authentication Ontology 53](#_Toc194236933)

[9.3 Search Ontology 54](#_Toc194236934)

[10. References 56](#_Toc194236935)

# INTRODUCTION OF THE THESIS

## Introduction- Motives and Goals

From the very beginning computer systems have evolved from centralized computing devices supporting static applications, into client-server environments that allow complex forms of distributed computing. Throughout this evolution the emphasize was always on data mobility instead of code mobility e.g., the client fetches desired information from the server by sending request to the server. However, a new phase of evolution is now under way that allows the mobility of code instead of data to gain some performance goals that will be described later.

The catalysts for this evolutionary path are mobile software agents – programs that are goal-directed and capable of suspending their execution on one platform and moving to another platform where they resume execution. More precisely, a software agent is a program that can exercise an individual's or organization's authority, work autonomously toward a goal, and meet and interact with other agents. Possible interactions among agents include contract and service negotiation, auctioning, and bargaining. This work focuses mainly on the security issues that arise when mobile agents collaborate with each other during their life time [1].

The advantage of collaborating multi-agents system alleviates the difficulty of gathering information throughout the networks from a variety of different data sources. An agent being a social entity collaborates with other agents to fulfill its goals. This collaboration of multiple agents brings several advantages. First, the operation is distributed to enhance system performance and information availability. Second, it eliminates the single point of failure as opposed to a single agent method. Third, the throughput is increased through parallelism.

However, in spite of all these benefits, a *non secure* collaboration among multiple agents could produce several security loopholes like data confidentiality and integrity during message passing, masquerading, access control, authentication of origin, agent’s baggage (information carried by the agent) protection, and repudiation attacks, etc. The task of secure mobile agents’ collaboration becomes more complicated when an agent replicates itself or delegates its privileges to some other agent.

The second problem in this context is that since agents need to communicate with each other, there must be a predefined standard and format to define their communication, i.e. they must use common language that will be based on their shared vocabulary.

Thus, the goal of this project is to describe a standard for the messages to communicate between different agents, to find out some potential security solutions to produce a secure collaborative environment, and finally, for the proof of concepts, implement the proposed solution using Java.

## Statement of the problem

The major problem of building a secure collaborative environment for a multi agents system can be subdivided into the following four aspects:

### Defining Message formats

The goal of this problem is to identify the way through which mobile agents can communicate with each other. For this purpose, existing standards for communication, like Mobile Agent System Interoperability Facility (MASIF) and Foundation for Intelligent Physical Agents (FIPA), will be critically studied.

### Identify potential threats

Identify potential threats involved during the collaboration. Some of them are described in section 1. For example, agents usually build agent societies where reputation and role of an agent is very important. Masquerading as another agent may harm the network resources and both agents, the agent that is being deceived and the agent whose identity has been captured. Open research issues are: how to secure an agent from another malicious agent during collaboration, how to secure an agent from another agent that tries to access agent resources or sensitive information, etc.

### Building Solution

After identifying potential threats, the next task will be to find out potential solution to overcome these threats. This will be a research work and will be based on some cryptographic algorithms for authentication, data confidentiality and integrity etc.

### Implementation

The final task will be to implement the solution proposed in the previous step. This implementation will be done using Java.

## The Goals of the Project

As described above, the summarized list of goals of this project is described below:

1. Describe/choose a suitable standard for mobile agent’s collaboration.
2. Identify potential security vulnerabilities related to mobile agent’s collaboration.
3. Design a solution for a secure collaborative environment for a multi agent system.
4. Implement the proposed solution using Java.

## The Road Map

The following steps were followed during the project:

* + 1. Study of the relevant papers, standards, technology for the background knowledge.
    2. Identify security vulnerabilities and threats in the collaboration environment of a multi agent system.
    3. Design suitable solution by following the relevant standards.
    4. Implement the solution in Java.
    5. Make final conclusions about what has been achieved during the thesis and suggest some future extensions.
    6. Write the report about thesis.
    7. Prepare to demonstrate.

## Relevant Standards and Background Material

As I discussed in the problem statement, the first step is to design or suggest the communication format. The proposed communication model should be flexible enough to support interoperability among different agent platforms. The following two standards were perceived to be significant in my problem area.

1. Mobile Agent System Interoperability Facility (MASIF): This is the first mobile agent standard presented by Object Management Group (OMG). It enables interoperability between agent platforms of different vendors. It integrates Remote Procedure Call (RPC) paradigm and Multi Agent technology.
2. Foundation for Intelligent Physical Agents (FIPA): FIPA is an international organization that is dedicated to promoting the industry of intelligent agents by openly developing specifications supporting interoperability among agents. The following FIPA specifications are helpful.
   1. *FIPA ACL (Agent Communication Language) Message Structure Specification:* A FIPA ACL message contains a set of one or more message parameters for negotiation among agents. This specification describes the format of these parameters e.g., message sender, receiver, and content parameter (for data to be exchanged) etc.
   2. *FIPA Ontology Service Specification:* FIPA communication is based on the assumption that two agents who wish to communicate share a common ontology. It ensures that the agents assign the same meaning to symbols used in the message. This specification helps us to build the agents that access a particular ontology service, agents that provide this service (known as Ontology Agents (OA)), and agents able to negotiate at run time a shared ontology for communication.
3. PKCS 7:

The Public-Key Cryptography Standards (PKCS) are offered by RSA Laboratories to promote the development of secure applications and other standards based on public-key cryptography. First published in 1991, PKCS have become widely implemented and referenced.

PKCS #7 specifies the syntax for any cryptographic data, such as a digital signature or a digital envelope, that is associated with a message. The purpose of PKCS #7 is to provide a standard syntax and a platform-independent digital representation of the cryptographic data. Applications that conform to the standard can interoperate regardless of platform differences, because the standard prescribes a way for them to read and compose messages.

PKCS #7, “Cryptographic Message Syntax Standard,” provides a particular example of the PKCS process at work. PKCS #7, now at version 1.5, defines the syntax for several types of cryptographically protected messages, including encrypted messages and messages with digital signatures. Originally an outgrowth of Internet Privacy-Enhanced Mail, PKCS #7 has become the basis for the now widely implemented S/MIME secure electronic mail specification. But its applications have not been limited to mail; PKCS #7 has also become a basis for message security in systems as diverse as the Secure Electronic Transaction (SET) specifications for bank card payments.

For the details about articles and different papers studied for the background and solution investigation purpose, please see the list of references.

## Scope of the Problem Area

The following major steps constitute my problem area,

* 1. The construction of messages including header and content information according to the chosen standard.
  2. Source and destination integrity: Both sender and receiver must be authenticated before the communication takes place.
  3. Massage confidentiality and integrity should be maintained during the communication.

Note: The cascading delegation (when an agent delegates its access rights to other agent), authorization, and message flooding (an agent sends infinite number of messages towards the victim agent to unnecessarily engage it or for the Denial of Service (DoS) attack) problems will be included in the future extensions.

## Schedule

(October 2007) – Preliminary background study.

(November 1, 2007 to December 15, 2007) – Study on Java programming and Java security

(December 15, 2007 to February 15, 2008) – Designing solution and Implementation.

(Feb 15, 2008 to March 15, 2008) –Wring report, preparing for the demonstration.

# BACKGROUND, RELEVANT STANDARDS AND TECHNOLOGIES

## Why Study Standards and Technologies

This chapter is dedicated to relevant standards and technologies that will help us to build a suitable background necessary to discuss the issues and build a vocabulary of terminologies used in the problem area. By studying these standards and technologies, we will be able to design a proper solution that will be acceptable to the variety of different vendors using different platforms for solution development for the same problem.

The solution which is based on currently accepted standards alleviates the interoperability issues and brings simplicity in the task of future extensions to the solution. In addition, standard guides us to follow a predefined path to solve the problem.

Similarly, technology allows us what we can or can’t do. Consequently, while designing a particular solution, we have to make sure that either it could be implemented using underlying technology or not. Another important aspect is the knowledge of related concepts and vocabulary of the technology and tools to be used before we are going to implement the solution. Here is the brief discussion of such standards and technologies.

## About Mobile Agents

This section describes mobile agents, their team organization, and structures. In any type of team structure, there are two types of communication between mobile agents: inter-team communication (communicating agents belongs to the same team) and intra-team communication (communicating agents belong to different teams). It should be noted that there is higher level of trust between agents when they are communicating with their team members as compared to the intra team communication. Hence, there will be less security requirements for inter-team communication, unlike intra-team communication.

### What is Mobile Agent?

An agent is authorized to perform a certain task on behalf of another. Agents possess the characteristics of delegacy (authority to act autonomously on behalf of another), competency (capability to accomplish the assigned task), and amenability (the ability to adapt behavior in order to increase the performance and effectively accomplish the required goal).

Mobile agents are the special category of agents that are also known as traveling agents. These agents, on behalf of user or organization, are capable to migrate from one agent platform (the place where agents perform their execution) to another, can suspend and resume their execution on another platform, and finally report the results of assigned task to their originator at home platform. This often improves performance by moving the agents to where the data reside instead of moving the data to where the agents reside. The alternative typical operation involves a client-server model. In this case, the agent, in the role of the client, requests that the server transmits volumes of data back to the agent to be analyzed. Oftentimes data must be returned by the agent to the server in a processed form. Significant bandwidth performance improvements can be achieved by running the agents within the same framework as the data. Mobile agent frameworks are currently rare due to the high level of trust required to accept a foreign agent onto one's data server. With advances in technologies for accountability and immunity, mobile agent systems are expected to become more popular.

This work is dedicated to build a secure communication model to support collaboration among different mobile agents. Therefore, in the preceding sections, I will use the term agent to refer to a mobile agent despite the fact that there is a difference between these two terms.

As mentioned in [5], in a multi agents’ system, agents work together in the form of agents’ team. The following sections describe the team organization and structure that has been devised by the current research in the area of multi agents’ systems.

### Mobile Agents Team Organization

The team formation depends upon several parameters e.g., goal to be achieved, the capability of individual agent, etc. This task becomes more complicated when you have to build teams dynamically, the description is not provided here, since it is beyond the scope of this work. However, a general definition of a valid team is as follows:

“A valid agents’ team is a set of agents M = {*a*i} that includes a connected sub-graph of the agent’s social network and whose skill set {*σ*i} fulfills the skill requirement for task Tk.” [5]

Mobile agents team organization consists of a set of N agents A = { *a*1, *a*2,……, *a*n} shown in the form a graph G = {V, E}, where V is the set of vertices or nodes and E is the edge set. Each agent *a*i is located at the corresponding node Vi. So N = |V| (number of nodes in a graph.)



Figure2.1: Agents master/slave team model

Each agent can be in one of three states *s*i = {0, 1, 2}, which corresponds to *uncommitted*, *committed*, and *active* respectively. An uncommitted agent is available and not assigned to any task. A *committed* agent is assigned a task, but the full team to work on the specific task has not been formed yet. An active agent is the member of a team that has fulfilled all of the skill requirements for a task and is actively working on that task. Active agents become uncommitted when the task is completed. Committed agents become uncommitted if the advertisement duration is reached without fulfilling the skill requirements for a given task.

The agents are also assigned a single skill *σ*= {1, ∑}, where ∑ is a number of different types of skills that are present within a specific environment. The edge set E restricts the interactions of agents.

### Agents Team Models

Another important issue in the context of team formation is the distribution of knowledge and autonomy level of each team member in the team. The following two types of team models are presented in [4]:

#### Centralized Knowledge Concept (CKC)

This type of team oriented concept consists of master/slave mobile agents. Initial request to perform a main task is submitted to the master agent. The master agent then creates a set of slave agents, prepares a team plan, and sends them to different hosts to perform individual tasks. Hence, in a CKC, all the knowledge of the team is concentrated towards the master agent.

#### Distributed Knowledge Concept (DKC)

In a DKC, all agents in a multi agent system posses the same level of knowledge. In this type of a team model, agents are more autonomous and have situation-specific knowledge to handle an unexpected situation. However, one agent can perform only one task.

## Foundation for Intelligent Physical Agents (FIPA)

The Foundation for Intelligent Physical Agents (FIPA) is an international organization that is dedicated to promoting the technology of intelligent agents by openly developing specifications supporting interoperability between agents and agent based applications. This occurs through open collaboration between its member organizations, which are companies and universities that are active in the field of agents. FIPA makes the results of its activities available to all interested parties and intends to contribute its results to the appropriate formal standard bodies where appropriate. The following discussion shows the efforts made by FIPA to standardize the agent’s communication and its security specifications.

### Agent Communication Language (ACL) Structure

A FIPA ACL message contains a set of one or more message parameters. Precisely which parameters are needed for effective agent communication will vary according to the situation; the only parameter that is mandatory in all ACL messages is the performative, although it is expected that most ACL messages will also contain sender, receiver and content parameters. If an agent does not recognize or is unable to process one or more of the parameters or parameter values, it can reply with the appropriate not-understood message. Specific implementations are free to include user-defined message parameters other than the FIPA ACL message parameters specified in Table 1. The semantics of these user-defined parameters is not defined by FIPA, and FIPA compliance does not require any particular interpretation of these parameters. The prefix string “X-” must be used for the names of these non-FIPA standard additional parameters [10]. The details of these parameters are given in Table 2.1.

Here, all other parameters except the content parameter, comprise the FIPA ACL message header containing useful information about sender, receiver, language, protocol, and ontology (will be discussed later in section 3.1.5) etc. The content parameter contains the actual data to be exchanged between two communicating agents. The values of this parameter are based upon the common ontology between the agents.

FIPA also permitted encodings and parameter orderings for ACL messages; these are described in other specifications. Each ACL message has precise syntax description encodings based on XML, text strings and several other schemes.

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| performative | Type of communicative acts |
| sender | Participant in communication |
| receiver | Participant in communication |
| reply-to | Participant in communication |
| content | Content of message |
| language | Description of Content |
| encoding | Description of Content |
| ontology | Description of Content |
| protocol | Control of conversation |
| conversation-id | Control of conversation |
| reply-with | Control of conversation |
| in-reply-to | Control of conversation |
| reply-by | Control of conversation |

Table 2.1: FIPA ACL message parameters

### Agent Security Management Specifications (FIPA ASMS)

The FIPA 98 Agent Security Management Specification (FIPA ASMS) introduces the concept of an Agent Platform Security Manager (APSM). Thus agents are required to communicate through APSM’s located at their respective hosts. The APSM communicates using FIPA ACL and is responsible for maintaining the platform security policy.

FIPA ASMS defines six potential security threats to multi agent system. These threats includes (1) disclosure of an agent’s private data, (2) unauthorized alteration of agent’s state/data, (3) illegal copy and replay of a message, (4) denial of service (to engage an agent by sending large number of messages), (5) repudiation, (6) spoofing and finally masquerading.

The FIPA agent security management model is shown in figure 2.2. Here is the short description of each component of that model:

#### Agent Management System (AMS)

As described in [9], the AMS is responsible for managing the administrative activities like creation/deletion of agents, registration of agents at the platform, and control over access to and use of the Agent Communication Channel (ACC).  The AMS can keep the agent’s credentials like key pairs in secure storage (e.g. hashed).

#### Agent Communication Channel (ACC)

This component is used for intra-platform communication between agents.

#### Internal Platform Message Transport

This component is used for communication between agents that are residing in the same platform.

#### Directory Facilitator (DF)

DF facilitates mobile agents to search for a particular agent that can offer the services they are intending to use. All the agents wishing to provide their services to other agents register themselves with the DF with agent-services parameters e.g., interaction-protocol, agent type, agent services, and ownership. FIPA ASMS introduces a new parameter called “Security-Context” for security specifications; it includes three types of information: agent certificate, owner certificate, and security encapsulation method supported by the agent. Both the ACC and AMS can register with a DF.  At a minimum, the AMS must register with the default DF of the platform.

#### Agent Platform Security Manager (APSM)

The APSM is responsible for maintaining platform and infrastructure security policies. The APSM is responsible for run-time activities, such as, communications, providing transport-level security, and creating audit trails. The APSM is responsible for enforcing the security policy of its domain, and can at its discretion, upgrade the level of security requested by an agent. The APSM cannot downgrade the level of services requested by an agent, but must inform the agent that the service level requested cannot be provided.

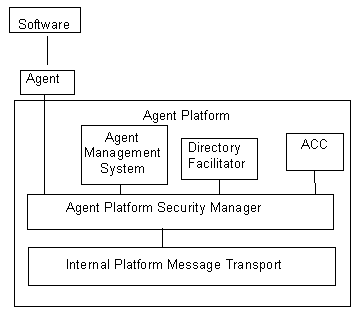


Figure2.2: The FIPA Agent Security Management Model [8]

### FIPA Agent Communication

Figure 2.3 shows the FIPA reference model for communication between two agents. In this reference model Message Transport Service (MTS) mediates between communicating agents to facilitate their communication on the underlying agent platform..

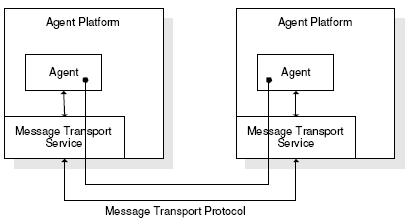


Figure 2.3: FIPA Agent Communication Reference Model

Other than inter-platform communication**,** MTS (with the consent of ACC) also provides intra-platform communication between agents residing on different agent platforms. FIPA defines the following three possible ways for intra-platform communication.

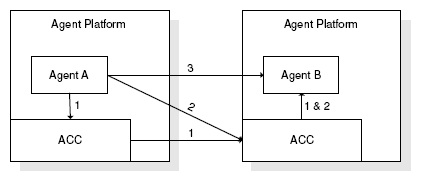
1. Agent A sends the message to its local ACC using an interface. The ACC then takes care of the transmission of the message to the correct remote ACC. The remote ACC will then eventually deliver the message to the intended remote agent B.
2. Agent A sends the message directly to the ACC running on the remote agent platform on which agent B resides. This remote ACC then delivers the message to B.
3.  Agent A sends the message directly to agent B, using a direct communication mechanism. The message transfer, including buffering of messages and any error messages, must be handled by the sending and receiving agents. No further specification of this communication mode is covered by FIPA.

Figure2.4: Intra Platform communication between mobile agents

### FIPA Agent Communication Security

In orderto provide transport level security for agents’ communication, FIPAintroduces the *envelope* parameter for an ACL message. The format of the envelope parameter is shown in figure 2.4. For details about the format of the envelope parameters, see [10].

**Level**

* 1. Low
  2. Medium
  3. High

**Mechanism**

1. DES-40
2. DES-56
3. AES
4. IDEA
5. RC2
6. RC4
7. RC5
8. RC6
9. Blowfish
10. CAST
11. SAFER
12. Proprietary

To: Destination Address

From: Return Address

**Confidentiality**

**Integrity**

**Authentication**

**Non-Repudiation**

**Mechanism**

1. MAC
2. SHA-1
3. MD-2
4. MD-4
5. MD-5
6. RIPEM
7. RIPEM-160
8. HMAC
9. Proprietary

**Mechanism**

1. DSA
2. DH Key Exch.
3. RSA
4. Kerberos
5. ECDSA
6. Proprietary

Figure 2.4: FIPA recommended format of the envelope parameter.

### FIPA Auditing

Finally FIPA describes how to maintain audit logs to provide accountability and non-repudiations. They describe the structure of logs and audit events. For more information about FIPA auditing specifications, please visit [10].

### FIPA Ontology Service Specification

One prerequisite for FIPA agent communication between agents is the presence of common ontology for the domain of interest. There are several issues related to the design of such ontologies e.g., what should be the structure, how and where the storage will be made, and how these ontologies will be fetched to share the information. FIPA describes all these issues in detail in its document [13].

Ontology Service Reference model provides a generic view about how ontologies are stored on different servers. These ontologies are fetched by the Ontology Agent (OA) using different protocols. OA then transfers the fetched information to different agents using Message Transport Service (MTS). This model also describes the interoperability issues and suggests the use of agent communication language and FIPA compliant OAs that will serve as an interface to FIPA compliant services.

FIPA ontology service specifications specify:

1. How to extend ontology. This is always a good engineering practice for future extension.
2. How to translate an ontology written in one language into another language.
3. How an OA agent can register with the Directory Facilitator (DF) and query to DF for the available ontology services.
4. About ontology classes and Open Knowledge Base Connectivity (OKBC) which supports an object-oriented representation of knowledge.

## Mobile Agent System Interoperability Facility (MASIF)

MASIF is a standard that defines the interoperability between different agent systems which are written in some language. MASIF standardize the following aspects:

1. Agent Management: Defines the way system administrator creates an agent, suspends, resumes, or terminates it.
2. Agent Transfer: Facilitate the migration of mobile agents from one platform to another.
3. Agent and Agent System Names: Helps agents and agent systems to identify each other.
4. Agent System Type and Location Syntax: Specify the types of agent systems and standardize their location syntax, so that agent systems can easily locate each other.

However, one major drawback is that MASIF doesn’t describe the communication between mobile agents. Since I am concentrating on agents´ communication security, therefore this standard is of less importance to me. However, if you are interested to study MASIF in detail, please check [9].

## Public Key Cryptographic Standard 7 (PKCS 7)

PKCS #7 specifies the syntax for any cryptographic data, such as a digital signature or a digital envelope that is associated with a message. The purpose of PKCS #7 is to provide a standard syntax and a platform-independent digital representation of the cryptographic data. Applications that conform to the standard can interoperate regardless of platform differences, because the standard prescribes a way for them to read and compose messages. PKCS #7 data message types include the following:

1. *ContentInfo:* Describes the most general type and defines the basic attributes and methods related to PKCS 7 message type. All other message types are derived from the ContentInfo data type.
2. *Data:* The data content type is just an octet string. The data content type is intended to refer to arbitrary octet strings, such as ASCII text files. This type refers to simple octet texts without any cryptographic operations applied to it.
3. *Signed Data:* The signed-data content type consists of content of any type and encrypted message digests of the content for zero or more signers. The encrypted digest for a signer is a "digital signature" on the content for that signer.
4. *Enveloped Data:* The enveloped-data content type consists of encrypted content of any type and encrypted content-encryption keys for one or more recipients. The combination of encrypted content and encrypted content-encryption key for a recipient is a "digital envelope" for that recipient.
5. *Signed and Enveloped Data Type:* The signed-and-enveloped-data content type consists of encrypted content of any type, encrypted content-encryption keys for one or more recipients, and doubly encrypted message digests for one or more signers. The "double encryption" consists of an encryption with a signer's private key followed by an encryption with the content-encryption key. [2]

The signed and enveloped data type will be used to provide data confidentiality, since it contains encrypted contents, message integrity with the help of digital signature of the message contents, and the origin integrity since it contains the digital signature signed by the private key of the signer and it can be verified by the recipient with the help of sender’s public key. The reader is encouraged to explore [2] for more details.

## About Technology

For the proof of solution design concepts, I built a prototype that provides the desired scenario of agents’ communication. The software and technology used as a platform and support to implement is briefly described below.

### Eclipse with Rich Client Platform (RCP) Plug-ins

The Eclipse platform is designed to serve as an open tools platform; it is designed in such a way that its components could be used to build any client application. The minimal set of plug-ins needed to build a rich client application is collectively known as the Rich Client Platform.

The Eclipse Platform is based on a mechanism for discovering, integrating, and running modules called *plug-ins*. A tool provider writes a tool as a separate plug-in that operates on files in the workspace and surfaces its tool-specific UI in the workbench. When the Platform is launched, the user is presented with an integrated development environment (IDE) composed of the set of available plug-ins. [14]

One major prerequisite for the Eclipse platform to run is the Java Virtual Machine (JVM); this support was provided by the Java Runtime Environment (JRE) version 1.5.

For further studies about Eclipse platform, please see [14].

### Scalable, Fault Tolerant Agent Grooming Environment (SAGE)

SAGE is an agent platform developed by National Institute of Information Technology (NIIT) (Rawalpindi, Islamabad, Pakistan) and Comtec (Japan). It is open source software and the aim of developing SAGE is to build a distributed decentralized, fault tolerant, scalable and lightweight agent platform according to new FIPA specifications. SAGE has several components, like Agent Management System (AMS), Directory Facilitator (DF), Message Transport Service, and Agent Communication Language (ACL). However, ACL is the component of my interest which will be used to create ACL messages that will be passed between agents during their communication. The details of ACL component will be explored in the next chapter. See [15] for further information about SAGE.

# THREATS AND SECURITY REQUIREMENTS

## Potential Threats

The threats related to communications between agents are not much different from the general communication of data between two parties. The following is short description of such threats. After describing these threats, the next goal is to identify security services that can counter each threat.

### Disclosure

A breach in the confidentiality of an agent's private data or meta-data. For example, an entity eavesdrops on the communication between agents and extracts information on the goals, plans, capabilities, etc. of these agents.

### Authentication of Source and Target

Both the identities of source and destination agents must be verified.

### Alteration

Unauthorized modification or corruption of an agent, its state, or data. For example, an Agent Communication Channel modifies the content of a message or a malicious agent attempts to modify the other agent’s private data.

### Copy and Replay

An attempt to copy a message and retransmit it. For example, a malicious agent creates an illegal copy of a message from an agent and retransmits it to its intended destination. This is a form of passive/active eavesdropping.

### Spoofing and Masquerading

An unauthorized agent claims the identity of another authorized or unauthorized agent. For example, an agent registers as a Directory Facilitator Agent and therefore receives information from other registering agents.

### Denial of Service (DoS)

An attack that attempts to deny resources to an agent.  For example, an agent floods the Directory Facilitator Agent with requests and the Directory Facilitator is unable to provide its services to other agents.

### Repudiation

An agent denies that it has received/sent a message or taken a specific action. For example, a commitment between two agents as the result of a contract negotiation is later ignored by one of the agents, the agent denies the negotiation has ever taken place and refuses to honor its part of the commitment.

## Security Requirements

The following is the abstract description of how agents’ communication will be safe from the prying eye. These security specifications set the preliminary grounds for the design of security system and eliminate the first five threats identified in the previous section. The last two threats i.e. DoS and Repudiation are treated to be beyond the scope of this work and will be discussed again in the future work with their possible solutions in mind.

### Authentication

To discourage the spoofing and masquerading activities, there is a need of strong authentication. Both agents who wish to exchange data will first authenticate each other to ensure that they are talking to the same agent as pretended by it. For this, there should be a mechanism of mutual authentication. The task of mutual authentication is usually achieved with the help of challenge response mechanism using nonce values sent as challenges. This will be described in the next chapter in detail.

### Data Confidentiality

To prevent information to be disclosed by unauthorized party, the information must be sent in encrypted form which will be in unreadable form. However, this requirement may be relaxed when both communicating agents belongs to the same team and they are residing in the same platform. In that case they can communicate through simple ACL messages and it will be responsibility of target platform to protect the exchanged information.

### Message Integrity

To detect possible modification of the message by an adversary, there must be a mechanism that could detect such unwilling modifications. Usually, the sender of the message generates a unique value for each message to be sent with the help of one way hash function. This value is known as message digest or hash. This hash is then encrypted to ensure its confidentiality and sent along with the encrypted message towards the receiver. The receiver then ensures the message integrity by calculating its own hash on the received message using the same function as used by the sender and then comparing both hash values. If both hash values are equal, then there will be no alteration in the message during communication.

### Source Integrity

Source and Recipient integrity (collectively known as origin integrity) means both sender and receiver should be verified by each other verifying that the data is sent by the same party by which it is communicating and not by the other malicious third party. To preserver origin integrity, the concept of digital signature is presented in which the sender of data digitally signs the data with help of its private key.

# DESIGN OF THE SECURITY SYSTEM

In the previous chapter, potential threats in the domain of our interest have been identified and their relevant security specifications were also described. Now, it’s the time to design a proper solution that should support:

* The standards described in chapter 2.
* The security services described in chapter 3.
* Can be implemented using the selected technology mentioned in chapter 2.
* Will be easily extensible in the future
* Can meet the interoperability issues, such that variety of other systems can interoperate with it.

## Creating FIPA Compliant ACL Messages

To construct FIPA compliant ACL messages, I used the ACL module of SAGE that is capable to create any ACL message template, header, and content information. For this purpose, an external acl.jar file was added in the project to use required engine classes.

### SAGE ACL Module Engine Classes

The following is the short description of these engine classes.

**1. ACLMessageInterface**

This class represents the FIPA (2000) compliant ACL Message as discussed in the FIPA 2000 specification (SC00061G). The purpose of this class is to provide developers with a high level class for creating ACL messages. This class contains code to create a template for ACL message and then get and set methods are used to retrieve and set the values for different parameters of ACL message discussed in section 2.3.1.

**2. SLTokenizer**

This class is responsible for encoding and decoding the content parameter of ACL message into a FIPA Compliant SL string. The encode method encodes a [[1]](#footnote-2)CFObject (into its corresponding FIPA compliant SL string using the corresponding Ontology. Similarly, decode method decodes the FIPA compliant SL string into [[2]](#footnote-3)CFContent object.

**3. CFAgentAction**

This class is basically a wrapper class or a facilitator class to hold an AgentAction. These objects are used by the parser and coder to transform these facilitators into a FIPA compliant message and vice versa. The purpose of CFAgentAction is to create ACL content parameter with the help of given ontology.

**4. ACLCodec**

This class contains actual methods to encode and decode the complete ACL message including ACL header and content parameter. These methods are static and can be invoked without creating the object of ACLCodec.

### Custom Designed Classes for Authentication Messages

Besides SAGE ACL engine classes, I also designed customized classes which are used to create ACL messages for the [[3]](#footnote-4)File Search application. These classes provide a generic view of the ACL message creation and behind the scene use the SAGE classes e.g. the description of classes used to create authentication messages is shown in figure 4.1.

ACL Message Factory class is more general class. It is responsible to create any kind of ACL message template and header with default parameter values including the content parameter. Authentication class extends ACLMsgFactory and contains data and methods that are used to create different ACL messages for mutual authentication between two agents. This class has authDC data item of type AuthenticationDC which is basically the storage object for authentication ontology data. The complete description of authentication ontology will be provided later. Please see the implementation chapter in order to study the detail description about other attributes and methods of these two classes.

ACL Message Factory

* aclMessage :ACLMessageInterface
* slt: SLTokenizer
* cfAgentAction: CFAgentAction
* aclContent: String
* aclCFContent: CFContent
* ACLMsgFactory(): Constructor
* createACLMsgWithDefaultParameters():ACLMessageInterface
* loadACLContentParameter(): void

Class Name

Attributes

Methods

Authentication

* sNonce:int
* rNonce:int
* authDC: AuthenticationDC
* Authentication(): Constructor
* generateChallenge():String
* receiveChallenge(String): void
* generateResponse(): String
* receiveResponse(String): void
* generateResponseAcknowledgement(): String
* receiveResponseAcknowledgement (String): void
* decodeAuthMsg(String): void

Figure 4.1: Authentication and ACL Message Factory classes’ hierarchy

### Example: Creating Simple ACL Message for Authentication Challenge

The process of creating ACL message with the help of the above mentioned classes can be best described with the help of an example. Here is the description of an ACL message for authentication that is used for sending challenge to supplicant in a challenge response authentication protocol.

Create Authentication Object

Generate nonce value

Set authentication ontology data object’s sender nonce value to nonce value generated

Create the ACL content parameter using CFAgentAction

Load the content parameter

Encode ACL message into String

SENDER

Receiver

Retrieve sNonce from current Auth object

Get sender nonce value from authentication ontology using CFAgentAction

Get the content parameter of ACL message

1. Create Authentication Object
2. Receive ACL message in String form and decode this into CFAgentaction

Set sNonce (sender nonce) of current Authentication object with received sender nonce

.

Figure 4.2: ACL message creation for sending and receiving authentication challenge

Figure 4.2 depicts the process of creating ACL message for authentication challenge. First, the sender instantiates an object of the Authentication class, generates nonce value, and sets AuthenticationDC (storage class for authentication ontology) object’s sNonce (sender nonce) with this nonce value. Then ACL message content parameter is created and loaded into ACL message and finally the entire ACL message is encoded into simple text.

On the other side, the receiver also creates an instance of Authentication object in order to decode ACL message and extract nonce value from it. It simply decodes the ACL message with the help of SLTokenizer, extracts content parameter from it in the form of CFAgentAction object. Finally, CFAgentAction is used to obtain the sender’s nonce value from the current AuthenticationOntology instance. The current authentication object’s sNonce is then set to the nonce value received.

## Authentication Protocol

The authentication protocol designed for agents’ identity verification is mutual authentication i.e. both communicating agents will authenticate each other. Suppose there are two agents, A and B, then the following must be ensured by each participant:

1. From the sender side A
   1. The identity of A and that the message was generated by A
   2. That the message was intended for B
   3. The integrity and originality of the message. It shouldn’t be sent multiple times to avoid copy and replay attack.
2. From the receiving side B
   1. The identity of B and that the reply message was generated by B
   2. That the message was intended for A
   3. The integrity and originality of the reply message

By keeping in mind the above requirements for mutual authentication, the following authentication protocol is designed.

1. A generates an ACL message containing randomly generated nonce NA into its content parameter. A then envelops this ACL message into [[4]](#footnote-5)PKCS 7 to ensure message confidentiality and integrity. Finally A sends PKCS 7 enveloped data to B as a challenge.
2. B receives the PKCS 7 enveloped challenge, opens it, decodes the ACL message, and extracts NA  from its content parameter. B then generates its own nonce NB, and sends PKCS 7 enveloped ACL message containing both NA (as response), and NB (as challenge) to A.
3. A, after receiving response from B, opens the ACL message, extracts both nonce values NA, and NB and performs the following operations
   1. Compares the NA received from B with the nonce value previously sent to B at step 1. **If** both nonce values match successfully **then** the identity of B is verified, since only B can open the PKCS 7 enveloped ACL message sent by A, **otherwise** Awill send authentication failure message to B.
   2. **If** Identity of B has been validated in step 3 (a), **then** A now has to prove its identity. For this purpose it sends back the nonce value NB (as a response) to B in the form of PKCS 7 enveloped data.
4. **If** Identity of B has been validated in step 3 (a), **then** B receives response from A, extracts NB from ACL message in the same fashion as in step 2 and checks that NB is the same nonce generated by it in step 2. **If** it is, **then** the identity of A is also verified, since only A can extract the message sent by B at step 2. B then **allows** A to start communication, otherwise the authentication has failed.

Challenge ()

Response ()

Challenge ()

[If Identity of B is validated] Response ()

[If Id. of B is not validated] authFail ()

[If res. is received from A] authResult ()

authA: Authentication

authB: Authentication

Figure 4.3: Sequence diagram of the authentication process between two Agents

Figure 4.3 shows the sequence of messages exchanged between two authenticating agents. Each agent first creates an Authentication object, and then participates in the authentication process by invoking methods of that object.

By carefully reading authentication protocol, a question strikes in mind that “What is the need to create ACL message and then envelop it using PKCS 7 functions to pass messages during authentication? Is it processing overhead?” The possible answer could be, Yes, It may be overhead for the processing, but we can achieve the following potential benefits by doing this.

1. *Generalization*: One advantage to use ACL messages for authentication is to treat authentication messages as other ordinary messages that will be exchanged between agents in the File Search application. So, irrespective of the message type, any message will be exchanged in the form of PKCS 7 enveloped ACL message. It will give more general view to overall application and reduces the overhead of producing authentication message separately.
2. *Source and Destination Integrity*: When we envelop the ACL message into PKCS 7, only the intended recipient can open that message. The source will be confident that only designated recipient can de-envelop the message ensuring recipient integrity (i.e. the message was intended for that recipient). Similarly, if recipient becomes successful to open the message, then sender integrity will be ensured, since PKCS 7 uses the mechanism of public key infrastructure and it will use sender’s public key to open that message. For example, PKCS 7 signed and enveloped data also contains encrypted digital signature signed by sender. By verifying this signature receiver can ensure sender integrity.
3. *Message Integrity*:  *A cure to Denial of Service Attack:* PKCS 7 signed data creates encrypted digest of the message and sends it along with the message. With the help of this digest, recipient can easily detect the possible modification in the message by the adversary in the middle. e.g., if the man-in-the-middle merely modifies the nonce value sent by the sender, then the response from the other party will not be verified by the sender. Although, adversary can’t masquerade in this case, but it can lead to denial of service and the authentication process will fail.
4. *Prevention of Replay Attack*: Since the message is also encrypted, it couldn’t be read by the man-in-the-middle. Hence message replay will be eliminated.

The next section describes how ACL messages are protected using PKCS 7 standard functions.

## PKCS 7 Message Enveloping

In order to provide message confidentiality, integrity, and origin integrity, I decided to use PKCS 7 standard functions to envelop ACL messages and then on the receiving end de-envelop those messages. A new class is designed named as PKCS\_7 that extends the original PKCS7 class which contains necessary functions to support PKCS 7 standard. The PKCS\_7 class just provides the stage to invoke the necessary functions from PKCS7 class. Figure 4.4 provides an overview of PKCS\_7 class, while Figure 4.5 depicts the PKCS 7 message enveloping and de-enveloping process.

Since PKCS 7 signed and enveloped data provides encrypted data of the message for confidentiality, encrypted digest for message integrity, and encrypted digital signature for the origin integrity, PKCS 7 signed and enveloped data type contains all the security services we need to protect an ACL message. The details about parameters of PKCS7 functions are provided in the implementation chapter.

Figure 4.4: PKCS\_7 class

PKCS7

PKCS\_7

* subject: DistinguishedName
* issuer: DistinguishedName
* recipient: DistinguishedName[]
* workingDir: String
* PKCs\_7(): Constructor
* envelopPKCS7 (String): String
* deEnvelopPKCS7 (String): String

PKCS Enveloped Message

ACL Message

Sender

Receiver

Open PKCS Enveloped Message

ACL Message

Figure 4.5: PKCS 7 message enveloping and de-enveloping

## Data Communication and Code Mobility

After defining the format of messages and providing required security services mentioned in chapter 3, the next step is to determine:

1. How to establish connection between local host and remote host?
2. How these messages will be transmitted from sender to receiver?
3. How code mobility would be achieved, i.e. how an agent (a piece of code) will migrate from one host to another.

First of all in order to establish connection between local host and remote host, an object of SocketAction is created at local host. When this object is instantiated, it invokes the parameterized constructor of SocketAction which takes an object of Socket type. This socket object is created by invoking parameterized constructor of the Socket which takes the remote host address or name and the port number of the remote host. The sample code for the above discussion to establish connection can be written as.

SockAction sockAction = **new** SocketAction (**new** Socket (130.237.239.162, 8080);

When socketAction object is instantiated, the Server instance (which is continuously listening for the connection request) accepts this connection request and establishes connection with requesting host. The Server instance after establishing connection invokes a separate thread ServerConnectionThread that serves the further requests from the connected host.

Class hierarchy of the classes that participate in establishing the connection between two hosts is shown in figure 4.6. The detail description of these classes will be provided in the implementation chapter. Figure 4.7 shows the sequence of messages that has been passed during the connection establishment.

Now after establishing the connection between two hosts, the string data can be easily exchanged between them with the help of send() and receive() methods of SocketAction. Similarly, if we have to migrate an agent (that is simple an object in our scenario), it is simply achieved with the help of sendObject() and receiveObject() methods of the same class. These methods simply send/receive an object with the help of object serialization and de-serialization. One thing should be noted here is that these will be originally moving objects that can store their current state and later can restore their state at the remote host. This is limited code mobility and can’t be treated as mobile agents, since mobile agents are capable to suspend their execution on one host and further resume their execution on another host. The main reason to limit the code mobility is the unavailability of a suitable “Agent Management System” for an agent platform that will be responsible for agents’ migration.

Figure 4.6: Class hierarchy for establishing connection between two hosts

ServerConnectionThread

* server: Server
* ServerConnectionThread(): Constructor
* run (): void

SocketAction

* PORTNUM: int
* inStream:DataInputStream
* outStream:PrintStream
* socket: Socket
* SocketAction():Constructor
* run (): void
* send(): void
* receive(): String
* sendObject(): void
* receiveObject(): Object
* closeConnection(): void

Thread

Server

* PORTNUM: int
* port:ServerSocket
* Server():Constructor
* run (): void
* finalize: void

Figure 4.7: Sequence diagram of establishing connection between local host A and remote server.

A: HostA

ConnectWithServer ()

B: SocketAction

ConnectWithSpecifiedPort ()

C: Server

InvokeServerThread ()

D: ServerConThread

## File Search Application

To simulate the design concepts related to secure communication between agents, we decided to develop a simple application that uses frequent communication among different agents. This application searches for a particular text file at different remote hosts with the help of mobile agents.

First, a static Master Agent (MA) takes an input from the user as an overall goal. User gives the file name and the list of remote hosts’ IP addresses to search that file. MA, after receiving the goal from a user, creates different teams of mobile agents based on that goal. MA forms the teams in a way such that there is one team leader and several team members. The number of teams depends upon the number of remote hosts on which the search operation will be performed, since there is one to one mapping between teams and remote hosts. MA then distributes the overall goal among the teams as follows:

1. Assign to each team a unique remote host IP address.
2. Assign to each team’s team member the file name to search and the particular drive name of the remote host to search that file, e.g. team member A of team B has to search file (abc.txt) from D drive (portion of the physical disk) of remote host of IP address 130.237.239.162.

Figure 4.8: Team formation and goal setting by static agent

[File name. List of remote hosts]

[File name. Remote host name]

[File name. Remote host name]

 Static Agent

Team Lead



Team member



The MA next establishes connection with the required remote hosts and sends each team to their relevant remote host to perform the search operation. Now, the following messages will be involved during the global search operation.

* + 1. If the file is successfully found by team member, that member will send a message to its team leader with required information about that file e.g. the absolute path of the file on remote host or the contents of that file.
    2. Team leader after receiving the success message from its team member will broadcast a message to each team leader of other teams and to the MA that the required file has been found with the help of a [[5]](#footnote-6)messenger agent. After receiving this message, every team will stop working and will be destroyed. However, before broadcasting the message, it is necessary to be authenticated with each recipient. Note that the MA will finally report to the end user about the result.
    3. If the particular team member finishes its search operation and the required file is not found i.e. the search operation becomes unsuccessful, then this team member will report to its team leader that the required file is not found through an ACL message. This will be inter team communication and need not to be enveloped by PKCS 7 for the sake of simplicity and processing overhead.
    4. On the other side, team leader will wait for the reporting by each team member. If all the team members report the failure of the search operation, it means that the required file is not found on the target platform. In this situation, team leader will send message (again with the help of messenger agent) to MA reporting that the required file is not found on the specified host platform.
    5. Similarly, if every team leader reports the failure of search operation to MA, MA then will report to the end user about the failure of message.

Figure 4.9 shows the hierarchical structure of the classes used to implement different agents in the file search application. Please see the implementation chapter to know the implementation details of these classes.

Figure 4.9: Hierarchy of the classes used to implement different agents.

Agent

MasterAgent

Team Lead

Team Member

Mob Agent Team

Messenger Agent

## Ontology Design

Gruber in 1993 stated that, *“The ontology is a formal explicit specification of a shared conceptualization.”* Further in 1995 Russel and Norving linked ontology with the Artificial Intelligence as,

*“Ontology is the formal description of the concepts and relations which can exist in the community of agents.”*

Hence the ontology is a common, shared, and formal description of important concepts in the specific domain.

Why would someone want to develop an ontology? Some of the reasons are:

* + - 1. To share common understanding of the structure of information among people or software agents
      2. To enable reuse of domain knowledge
      3. To make domain assumptions explicit
      4. To separate domain knowledge from the operational knowledge
      5. To analyze domain knowledge [17]

The following figure shows the hierarchy of classes that has been designed to build the ontology for our File Search Application.

AUTHENTICATION

ONTOLOGY

TEAM GOAL

INDIVIDUAL GOAL

GOAL

SEARCH

SUCCESS

FAILURE

Figure 4.9: Hierarchy of the ontology classes designed for File Search Application

Here is the short description of the class hierarchy shown in above diagram. The related data items are written within small brackets while the complete description about these data items and the implementation of the ontology is mentioned in the next chapter.

1. ONTOLOGY (Name: String)
2. GOAL (File Name: String, List of Remote Hosts: String Array)
   1. TEAM GOAL (Remote Host Address: String)
   2. INDIVIDUAL GOAL (Remote Host Drive Name: String)
3. SEARCH (Search Result: Boolean)
   1. SUCCESS (File Contents: String, Remote Host Address: String)
   2. FALILURE (Description: String)
4. AUTHENTICATION (Sender Nonce: Integer, Receiver Nonce: Integer)

# IMPLEMENTATION

The technology, tools and pre-built plug-ins are described in section 2.5 and 2.6 in chapter 2. In this chapter we will explore the implementation details of the concepts described in the previous chapter. Following is the categorical description of different types that have been developed for our File Search application.

## Creating ACL Messages

Below are the custom designed classes to create different types of ACL messages involved in the File Search Application.

### ACL Message Factory Class (ACLMsgFactory.java)

This is more general class. The class is responsible for creation of any kind of ACL message template and header with default parameter values except the content parameter. See Appendix A for further implementation details.

### Search Result ACL Messages Class (SearchResulACLMessage.java)

This class is used to generate different type of ACL messages used in File Search Application e.g. search success message, search failure message etc. See Appendix A for more implementation details.

## Implementation of Authentication Protocol and Authentication Class (Authentication.java)

This class extends ACLMsgFactory. It contains the data and methods that are used to create different ACL messages for mutual authentication between two agents. The complete description of the challenge response authentication protocol will be described in next section. See Appendix B for implementation details.

## PKCS 7 Implementation (PKCS\_7.java)

This class is extended by original PKCS7 class that has been provided in the plugin. The major responsibility of this class is to envelop and de-envelop the ACL messages according to PKCS 7 standard. PKCS signed and enveloped data provides message confidentiality, integrity, and origin integrity. Additional functionalities include converting a byte array into hexadecimal and ASCII strings which are used when enveloping and opening PKCS 7 enveloped data. The following is the description about the data members and methods of this class. See Appendix C for the implementation details

## Data Communication and Code Mobility Implementation

Three specialized classes are designed to support data transmission and code mobility. First, the client connects with the remote host running the Server class instance with the help of SocketAction object. Server object (which is listening continuously for the client request) connects with the client and invoke a separate thread defined in ServerConnectionThread class which then handles all the requests from the client. See the Appendix A for further implementation details.

## Agents Implementation in File Search Application

To implement the different type of agents involved in the File Search Application several classes are designed. See Appendix B for the implementation details.

## Ontology Implementation

An ontology with the help of SAGE ontology support classes is implemented with the help of three files. The first is an interface that defines data items included in the ontology. It is also known as vocabulary class e.g. GoalVocabulary. The second is an ontology class that extends the basic Ontology class from SAGE and implements the relevant vocabulary class. And, finally, the storage class, that stores, retrieves, and sets the data items included in the ontology. Here is the implementation code of three main ontologies included in our domain of interest. Due to limited space, only interface and ontology classes are shown. Storage classes contain only set and get methods to read or write data items values. The SAGE based java code for File Search Application is given in Appendix C.

# Conclusions and Future Work

This chapter winds up all the work done and concludes some crucial achievements and remaining problems and issues in the area of secure agents’ communication.

## Conclusions

Agent based programming is the new programming paradigm and faces lots of security challenges due to which it is less often used. One of the security vulnerabilities in this domain is “How to protect the information that is exchanged by communicating agents and how agents can collaborate securely?”

The first step in describing the communication of agents is to define the format of the messages that will be exchanged during communication. FIPA ACL messages have been selected for this purpose in order to choose a solution that will be interoperable between different types of agent systems. To implement FIPA compliant ACL messages, an open source SAGE provided by NIIT (Rawalpindi, Islamabad, Pakistan) is used.

The next step after choosing a standardized message format is to protect these messages during their exchange. Potential threats to solve during the communication of message are described in chapter 3.

Keeping in mind these possible threats, we decided to use the PKCS 7 standard for messages security. The major reasons to choose PKCS 7 standard to perform cryptographic operations for the messages protection include the interoperability issues, generally accepted solution, fulfillment of PKCS 7 with the security requirements described in chapter 3, and the reuse of PKCS 7 engine classes. So the PKCS 7 signed and enveloped data types were used to protect ACL messages. PKCS 7 signed and enveloped data includes encrypted hash of the message to maintain message integrity, encrypted digital signature to ensure the origin or source integrity of the message, and encrypted contents of the message to retain its confidentiality. Hence, a PKCS 7 based solution was designed in chapter 4 with the help of suitable class and sequence diagrams.

Before going to negotiate, agents are required to authenticate each other to ensure that they are communicating with the attended party and no third part is masquerading in between. To cope with this situation, two way authentication protocol is designed that authenticates both communicating entities.

Another important aspect is how to transmit PKCS 7 enveloped messages over the network towards target platform. For this, the concept of socket programming is used and a special class SocketAction is designed that is responsible for establishing the connection between two hosts and also transmitting text data and objects.

After clearing the entire design concepts, a sample application, known as File Search Application, is specified for the implementation whose description is given in chapter 4. This application uses the necessary communication between agents and is used to implement a prototype to show the desired results. Different types of classes are designed to play the roles for different agents involved in this application.

## Future Work

There is always a room for improvements. When this job was started, the several other security vulnerabilities were also included, but unfortunately, due to the lack of time, emphasize was placed only on the communication aspect. However, these deficiencies along with other that were noticed during this work are best elaborated here along with their possible solutions.

### Cascading Delegation Problem

The concept of secure cascaded delegation is very important in agent reproduction. Cascaded delegation occurs when one agent may authorize other agents to perform some operations under its original access rights. Secure delegation has the number of open research challenges like: reconstruction of the complete delegation chain and verification of proper authorization of an agent claiming on behalf of another agent. Open research issues are: how to secure an agent from another malicious agent during collaboration, how to secure an agent from another agent that tries to access agent resources or sensitive information, etc.

### Authorization Control

To exchange crucial information, and access the sensitive information from the target agent, authorization is the next step after authentication i.e. what is allowed and what is not allowed to the communicating agent, which information can be exchanged and which is not allowed. For this purpose, an authorization access control mechanism should be implemented to prevent unauthorized access of the agent’s baggage.

A potential solution to build the authorization control system is to use the Role Based Access Control (RBAC) model to implement access rights. RBAC defines access control model that relates roles, permissions, constraints and users (external entities). The detail description about RBAC is given in [17].

### Repudiation

Repudiation is the act to deny participating in communication after it has taken place. Current solution of the secure collaboration doesn’t address this problem. An agent may simply repudiate that it has obtain some particular information during collaboration.

The possible solution of the repudiation attack is to keep record of the communication and maintain the log with suitable digital proof such as digital signature etc. How potential questions in this respect are: Which entries should be included in the log file? What are the storage and retrieval mechanisms? How long a particular entry in the log should be kept available and the protection of log as well?

### Message Flooding

Message flooding is the form of attack in which the adversary sends an infinite number of messages to the victim in order to engage it unnecessarily. This type of attack often leads to Denial of Services attack.

This type of DoS attack can be avoided by keeping track of the sender address and if messages coming from a particular address exceed to the specific limit and these messages yields no fruitful results then that sender can be blocked or requests from that sender may simply be ignored.

# Appendix A: Creating, Enveloping and Sending ACL Messages

## ACL message Factory Class (ACLMessageFactory.java)

The following are the data members:

* 1. ACLMessageInterface aclMessage;
  2. Stores ACL message.
  3. SLTokenizer slt; Used to encode and decode ACL message content parameter.
  4. CFAgentAction cfAgentAction; Used to create ACL content parameter.
  5. String aclContent; Stores content in the form of string.
  6. CFContent aclCFContent; Stores ACL content in the form of CFContent.

The following are the methods:

1. **public** ACLMsgFactory(){…}; Default constructor creates ACL Message Template with default header information.
2. **Private** ACLMessageInterface createACLMsgWithDefaultParameters(){…}; Create ACL message with default parameters’ values except content parameter.
3. **private** **void** loadACLContentParameter(){…}; Load the current ACL content parameter.

## File Search Result ACL Messages (SearchResultACLMessage.java)

**package** com.magnet.securecollaboration.application;

**import** org.sage.core.acl.aclcodec.ACLCodec;

**import** org.sage.core.acl.cf.CFAgentAction;

**import** org.sage.core.acl.cf.CFPrimitive;

**import** org.sage.core.acl.sl.codec.SLTokenizer;

**import** com.magnet.securecollaboration.tests.PKCS\_7;

//-- This class contains methods which are used to inform

// about search results.

**public** **class** SearchResultACLMessage **extends** ACLMsgFactory{

//-- private data members--

**public** **boolean** searchResult; // represents search success or failure

**public** String hostName;// remote host name.

**private** SearchDC searchDC;

**private** PKCS\_7 pkcs7;

String envData; //-- contains the pkcs7 enveloped data

String openedData;//-- contains de-enveloped data

//-- constructor, create ACL msg template and instantiate other data members

**public** SearchResultACLMessage() {

**super**(); //-- create an ACL message with default parameter values.

pkcs7 = **new** PKCS\_7(); //-- Instantiate PKCS\_7 object

envData = ""; openedData = "";//-- initialize.

//-- get an instance of SLTokenizer and set the ontology.

slt = SLTokenizer.*getInstance*();

slt.setOntology(SearchOntology.*getInstance*());

//-- set the ontology parameter for ACL msg

aclMessage.setOntology(SearchOntology.*NAME*);

searchDC = **new** SearchDC(); //-- instantiate the searchDC

//-- instantiate cfAgentAction with Search ontology

cfAgentAction = **new** CFAgentAction(SearchOntology.*NAME*);**}**

//-- Generate ACL Search successfull message

//-- Return PKCS7 enveloped ACL meesage

**private** String generateSuccessMsg(String fileContents, String n remoteHostAddress){

//-- Set the search result

**this**.searchDC.setSearchResult(**true**);

**this**.searchDC.setFileContents(fileContents);

**this**.searchDC.setRemoteHostAddress(remoteHostAddress);

//-- create the content parameter for ACL msg.

**this**.cfAgentAction.set(SearchOntology.*SEARCH\_RESULT*,CFPrimitive.*getCFPrimitiveFor*(searchDC.getSearchResult()));

**this**.cfAgentAction.set(SearchOntology.*FILE\_CONTENTS*,CFPrimitive.*getCFPrimitiveFor*(searchDC.getFileContents()));

**this**.cfAgentAction.set(SearchOntology.*REMOTE\_HOST\_ADDRESS*,CFPrimitive.*getCFPrimitiveFor*(searchDC.getRemoteHostAddress()));

//-- load ACL content parameter

aclMessage = **super**.loadACLContentParameter(aclMessage, slt, cfAgentAction);

//--Envelop ACL msg into PKCS 7

envData = pkcs7.envelopPKCS7(aclMessage.toString());

**return** envData; //--Send PKCS 7 enveloped data **}**

**private** String generateFailureMsg(String description){

//-- Set the search result

**this**.searchDC.setSearchResult(**false**);

**this**.searchDC.setFailureDescription(description);

//-- create the content parameter for ACL msg.

**this**.cfAgentAction.set(SearchOntology.*SEARCH\_RESULT*, CFPrimitive.*getCFPrimitiveFor*(searchDC.getSearchResult()));

**this**.cfAgentAction.set(SearchOntology.*DESCRIPTION*,CFPrimitive.*getCFPrimitiveFor*(searchDC.getFailureDescription()));

//-- load ACL content parameter

aclMessage = **super**.loadACLContentParameter(aclMessage, slt, cfAgentAction);

//--Envelop ACL msg into PKCS 7

envData = pkcs7.envelopPKCS7(aclMessage.toString());

**return** envData; //--Send PKCS 7 enveloped data}

//-- This method just opens the PKCS 7 enveloped success/failure message.

**private** **void** openSearchResultMessage(String pkcsEnvSearchResultMsg){

//--Open PKCS 7 data

openedData = pkcs7.deEnvelopPKCS7(pkcsEnvSearchResultMsg);

//--Decode ACL msg

**this**.decodeSearchResultMsg(openedData);

//--Pick up and set the value of search result variable.

searchResult = cfAgentAction.getBoolean(SearchOntology.*SEARCH\_RESULT*);

}

//-- decode any ACL msg related to search result.

**private** **void** decodeSearchResultMsg(String aclMessage) {

// ----------------Decode the ACL message----------------------

**this**.aclMessage = ACLCodec.*decode*(aclMessage);

// --- Get an instance of SLTokenizer that will decode the content parameter of the ACL message---

**this**.slt = SLTokenizer.*getInstance*();

**try** {

**this**.aclCFContent = slt.decode(**this**.aclMessage.getContent(),

SearchOntology.*getInstance*());

}

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

**this**.cfAgentAction = (CFAgentAction)aclCFContent.getObject();

}

## Authentication.java

The following are the data members:

1. **int** sNonce; Contains sender nonce value
2. **int** sNonce; Contains receiver nonce value
3. AuthenticationDC authDC; Object that stores authentication ontology data.

The following are the methods:

1. **public** Authentication(){…}; Constructor, creates ACL message template as follows
   * 1. Get the instance of SLTokenizer and sets it ontology with AuthenticationOntology (Ontology designed for authentication).
     2. Also set the ontology name for ACL message with AuthenticationOntology name.
     3. Instantiate the AuthenticationDC object. The object that stores data related to AuthenticationOntology.
     4. Also instantiate CFAgentAction object with its parameterized constructor with parameter value of AuthenticationOntology name.
2. **public** String generateChallenge(){…}; Generates an ACL message that contains challenge and envelop that ACL message into PKCS 7.
3. **public** void receiveChallenge(String pkcsEnvChallenge) {…};
   * 1. Receive PKCS 7 enveloped challenge.
     2. Open PKCS 7 data.
     3. Check the confidentiality and integrity of the data received.
     4. Decode ACL message, extract sender nonce and set the sNonce of the current Authentication object with the received nonce.
4. **public** String generateResponse(){…}; Generates an ACL message that contains response to the challenge from sender and envelop that ACL message into PKCS 7.
5. **public** void receiveResponse(String pkcsEnvChallenge) {…};
   * 1. Receive PKCS 7 enveloped response.
     2. Open PKCS 7 data.
     3. Check the confidentiality and integrity of the data received.
     4. Decode ACL message, extract both nonce values (sender and receiver)
     5. Verify the sNonce (sender nonce) sent back by supplicant.
     6. Set the rNonce of the current Authentication object with the received nonce as challenge.
6. **public** String generateResponseAcknowledgement(){…}; Create an ACL message containing nonce value received from supplicant as challenge. Envelop ACL message into PKCS 7.
7. **public** **boolean** receiveResponseAcknowledgement (String pkcsEnvChallenge) {…};
   * 1. Receive PKCS 7 enveloped response.
     2. Open PKCS 7 data.
     3. Check the confidentiality and integrity of the data received.
     4. Decode ACL message, extract both nonce values (sender and receiver)
     5. Verify the rNonce (receiver nonce) sent back by initial authenticator.
     6. Intimate authentication result.
8. **private** **void** decodeAuthMsg(String aclMessage) {…}; Decode any ACL message related to mutual authentication and extract content parameter from it.

## PKCS\_7.java

The following are the data members:

1. [[6]](#footnote-7)DistinguishedName subject; Contains globally unique name for the subject, the certificate holder.
2. DistinguishedName issuer; Contains globally unique name for the issuer, the certificate issuer.
3. DistinguishedName[] recipient; Contains the list of globally unique name(s) for the recipient(s).
4. ASN1Encoding req\_ASN1Cert; Requested certificate of type ASN1Encoding.
5. String workingDir; Contains certificate repository path on the system.

The following are the methods:

1. **public** PKCS\_7() {...}; Constructor, instantiate the PKCSobject.
2. **Public** StingenvelopPKCS7 (String aclMessage) {…};This method envelops ACL message and creates a digital envelop that is secure
   * 1. With respect to message confidentiality, integrity, and origin integrity.
     2. creates signed and enveloped data with encrypted data contents, encrypted digital signature, encrypted digested value
3. **Public** StingdeEnvelopPKCS7 (String pkcs7EnvData) {…}; This method de-envelops PKCS 7 enveloped ACL message.

## Server.java

Data members:

* 1. **public** **static** **final** **int** *PORTNUM* = 1234; Port number of the remote host to connect with.
  2. **private** ServerSocket port;
  3. **private** Socket clientSocket;//-- take a client socket object--

Methods:

1. **public** Server() {…}; Try to grab the specified port with the server socket.
2. **public** **void** run() {…}; Checks for the server port, wait until client requests for the connection, connect with the client and invoke a separate thread to serve the client for further requests.
3. **public** **void** finalize() {…}; Closes the port

## SocketAction.java

Data members:

1. **public** **static** **final** **int** *PORT* = 1234;
2. **private** DataInputStream inStream;
3. **protected** PrintStream outStream;
4. **private** Socket socket;

Methods:

1. **public** SocketAction() {…}; Implicit constructor does nothing.
2. **public** SocketAction(Socket sock) {…}; Initialize the Data Input/Output steams and the socket object.
3. **public** **void** run() {..}; SocketAction thread also does nothing
4. **public** **void** send(String s) {…}; Send the string data to connect remote host
5. **public** **void** sendObject(Object objToSend) **throws** IOException {…}; Send any type of object to the connected remote host.
6. **public** String receive() {}; Receive string data from connect remote host
7. **public** Object receiveObject() **throws** IOException,ClassNotFoundException {…}; Receive any type of object from connect remote host
8. **public** **void** closeConnections() {…}; Closes connection with remote host.
9. **public** **boolean** isConnected() {…}; Check whether the connection is established or not.

## ServerConnectionThread.java

Data members:

* 1. **private** Server server = **null**; Contains the reference of server object that invoke this thread.

Methods:

1. **public** ServerConnectionThread(){…};Implicit constructor does nothing.
2. **public** ServerConnectionThread(Server server, Socket sock) {…}; Explicit constructor, invokes the constructor of SocketAction and initialize the socket object. It also initiale the server object to the server that invoke this thread.
3. **public** **void** run () {**…**}; Contains the actual code to handle all client requests by the server.

# Appendix B: Agents Implementation In File Search Application

### Agent Class Agent.java)

1. **public** **long** agentID\_; Stores agent ID.
2. **public** Agent() {…}; Initialize agent ID with random number.
3. **public** Agent(**long** agentID) {…}; Initialize agent ID with random number.

### Master Agent Class (MasterAgent.java)

This class represents master agent which takes input (goal, remote host info) from a user and creates teams of mobile agents, assigns them a goal and sends them to different remote hosts. Team leaders will give the feedback to master agent and, finally, master agent will report to the user about the search results.

Data members:

1. **public** MATeam[] maTeams; Teams of Mobile Agents
2. **public** GoalDC goal; Overall goal

Methods:

1. **public** MasterAgent(){…};Constructor, set the agent ID and invoke the getUserInput method.
2. **public** **void** getUserInput() **throws** IOException {…}; Get input from user.
3. **public** **void** createTeams(){…}; Creates teams of mobile agent (centralized team model) according to the goal received.
4. **public** **void** getUserInput(String searchResult){…}; Report the end result to user.
5. **public** **void** listenFeedback(){};Continuously listen for the feedback from team members.

### Messenger Agent Class (Messenger.java)

Private data members:

1. **private** String pkcs7EnvMessage; Contains the message to be sent.

Methods:

1. **public** **void** setMessage(String msg){…}; Set the value of message to send
2. **public** String setMessage(){…};Get the message from messenger

### Mobile Agents Team Class (MATeam.java)

Data Members:

1. **public** TeamLead teamLead;
2. **public** TeamMember[] teamMember;
3. **public** GoalDC goal; Represents the overall goal.

Methods:

1. **public** MATeam() {…}; Constructor, create and instantiate team leader, also create and instantiate each team member and instantiate the team goal.

### Agent Team Lead Class (TeamLead.java)

Data Members:

1. **public** GoalDC teamGoal; // goal for individual team.
2. **public** TeamLead() {}

Methods:

1. **public** **void** listenFeedback(){…}; Continuously listen for the feedback from team members.
2. **public** **void** broadcastSuccessMsg(){…};Broadcast the success message to each team lead and the master agent.
3. **public** **void** sendFailureMsg(){…}; Sends the search failure message to the master agent.

### Agent Team Member Class (TeamMember.java)

Private members:

1. **public** GoalDC goal; Individual goal of mobile agent.
2. **public** TeamMember(){…}; Constructor, initialize individual goal object
3. **public** **boolean** searchFile(String fileName, String driveName){…}; This method searches for a particular file and return true or false depending on the search result.
4. **public** **void** sendSuccessMsg(){…};Sends the search success message to the team lead.
5. **public** **void** sendFailureMsg(){};Sends the search message message to the team lead.

# Appendix C: Ontology Implementation for File Search Application

## Goal Ontology

**public** **interface** GoalVocabulary {

**public** **static** **final** String *NAME* = "Secure-Collaboration";

// --parameters related to overall goal

**public** **static** **final** String *FILE\_NAME* = "File-Name";// file name to search

**public** **static** **final** String[] *HOST\_NAMES* = {"List-of-IPs-of-different-hosts"};

// --parameters related to team goal, will extend overall goal

**public** **static** **final** String *HOST\_NAME* = "Host-name-to-perform-search";

// --parameters related to individual team member goal, will extend team goal

**public** **static** **final** String *HOST\_DRIVE\_NAME* = "Host-drive-name";}

**public** **class** GoalOntology **extends** Ontology **implements** GoalVocabulary {

**private** **static** Ontology *theInstance* = **new** GoalOntology();

**public** **static** Ontology getInstance(){**return** *theInstance*;}

**public** GoalOntology(){

**super**(*NAME*, BasicOntology.*getInstance*());

**try**

{

//--- Set the ontology name------

AgentActionSchema cs = **new** AgentActionSchema(*NAME*);

//--- parameters for overall Goal------

cs.add(*FILE\_NAME*, (PrimitiveSchema) getSchema(BasicOntology.*STRING*));

//--- parameter(s) for team Goal------

cs.add(*HOST\_NAME*, (PrimitiveSchema) getSchema(BasicOntology.*STRING*));

//--- parameter(s) for individual Goal------

cs.add(*HOST\_DRIVE\_NAME*, (PrimitiveSchema) getSchema(BasicOntology.*STRING*));

**this**.add(cs);

}

**catch**(Exception exp)

{

exp.printStackTrace();

}

}

}

## Authentication Ontology

//-- This interface contains parameters needed for mutual authentication

//-- between sender and receiver.

**public** **interface** AuthenticationVocabulary {

//-- The name of ontology

**public** **static** **final** String *NAME* = "Authentication";

//1.--Nonce value by the sender--

**public** **static** **final** String *S\_NONCE* = "Sender-Nonce";

//2.--Nonce value by the receiver..

**public** **static** **final** String *R\_NONCE* = "Receiver-Nonce";}

**public** **class** AuthenticationOntology **extends** Ontology **implements** AuthenticationVocabulary {

**private** **static** Ontology *theInstance* = **new** AuthenticationOntology();

**public** **static** Ontology getInstance()

{

**return** *theInstance*;

}

**public** AuthenticationOntology()

{

**super**(*NAME*, BasicOntology.*getInstance*());

**try**

{

AgentActionSchema cs = **new** AgentActionSchema(*NAME*);

cs.add(*S\_NONCE*, (PrimitiveSchema) getSchema(BasicOntology.*INTEGER*));

cs.add(*R\_NONCE*, (PrimitiveSchema) getSchema(BasicOntology.*INTEGER*));

**this**.add(cs);

}

**catch**(Exception exp)

{

exp.printStackTrace();

} }}

## Search Ontology

**public** **interface** SearchVocabulary {

//1.--The Name--

**public** **static** **final** String *NAME* = "Search\_Result";

//2.--The Search Result--

**public** **static** **final** String *SEARCH\_RESULT* = "Search\_Result";

//-- i.SUCCESS (File Contents: String, Remote Host Address: String)

//--File contents..

**public** **static** **final** String *FILE\_CONTENTS* = "File-Contents";

//--Remote Host Address..

**public** **static** **final** String *REMOTE\_HOST\_ADDRESS* = "Remote-Host-Address";

//-- ii. FALILURE (Description: String)

//--Description about search failure.

**public** **static** **final** String *DESCRIPTION* = "Description";

}

**public** **class** SearchOntology **extends** Ontology **implements** SearchVocabulary {

**private** **static** Ontology *theInstance* = **new** SearchOntology();

**public** **static** Ontology getInstance(){ **return** *theInstance*;}

**public** SearchOntology()

{

**super**(*NAME*, BasicOntology.*getInstance*());

**try**

{

AgentActionSchema cs = **new** AgentActionSchema(*NAME*);

cs.add(*SEARCH\_RESULT*, (PrimitiveSchema) getSchema(BasicOntology.*BOOLEAN*));

cs.add(*REMOTE\_HOST\_ADDRESS*, (PrimitiveSchema) getSchema(BasicOntology.*STRING*));

cs.add(*FILE\_CONTENTS*, (PrimitiveSchema) getSchema(BasicOntology.*STRING*));

cs.add(*DESCRIPTION*, (PrimitiveSchema) getSchema(BasicOntology.*STRING*));

**this**.add(cs);

}

**catch**(Exception exp)

{

exp.printStackTrace();

}

}

}

# References

1. Wayne Jansen, Tom Karygiannis, National Institute of Standards and technology (NIST) Special Publication 800-19 – Mobile Agent Security
2. An RSA Laboratories Technical Note, PKCS #7: Cryptographic Message Syntax Standard Version 1.5, Revised November 1, 1993.
3. [David Wallace Croft](http://www.alumni.caltech.edu/~croft/), Intelligent Software Agents:Definitions and Applications, March 1997, URL: <http://alumnus.caltech.edu/~croft/research/agent/definition/>, Accessed: Feb 22, 08.
4. Neeran M. Karnik, Anand R. Tripathi, “Security in Aganta Mobile System”, Department of Computer Science, University of Minnesota, Minneapolis MN 55 455, U.S.A.
5. Yuh-Jong Hu, “Some thoughts on agent trust and delegation”, URL: http://portal.acm.org/citation.cfm?id=375735.376424.
6. Gordan Jezic, Mario Kusek, Sasa Desic, Ozren Labor, “Multi-agent System for Remote Software Operations”, URL: http://www.springerlink.com/content/txn6gr2h9h052bjt. March 26, 08.
7. Blazej Bulka, Mathew Gaston, “Local Strategy learning in networked multi-agent team formation”, URL: http://maple.cs.umbc.edu/papers/jaamas06-bulka.pdf. Date Accessed: March 26, 08.
8. Mathew E. Gaston, “Team Formation in complex networks”, URL: http://maple.cs.umbc.edu/papers/naacs03-gaston.pdf. Date Accessed: March 26, 08.
9. Dejan Milojicic, Markus Breugst, Ingo Busse, John Campbell, “MASIF, The OMG Mobile Agent System Interoperability Facility”, URL: http://www.hpl.hp.com/personal/Dejan\_Milojicic/ma4.pdf.
10. FIPA 98 Specification Part 10, Version 1.0, “Agent Security Management”, October 23, 1998.
11. Niklas Borselius and Chris J. Mitchell , “Securing FIPA agent communication”, Information Security Group, Royal Holloway, University of London.
12. FIPA Specifications, “FIPA ACL Message Structure.”, URL: http://www.fipa.org/specs/fipa00061/SC00061G.html.
13. FIPA Specifications, “FIPA Ontology Service Specifications”, URL: http://www.fipa.org/specs/fipa00086/XC00086D.pdf
14. Object Technology International Inc., “Eclipse Platform Technical Overview”, Revised Feb 2003
15. NIIT Open Source: Scalable, Fault Tolerant Agent Grooming Environment, URL: <http://sage.niit.edu.pk/SAGE_Components.htm>, Accessed: March 2, 2008.
16. Natalya F. Noy and Deborah L. McGuinness Stanford University, CA, 94305, “Ontology Development 101: A Guide to Creating Your First Ontology”, URL: [http://protege.stanford.edu/publications/ontology\_development/ontology101-noymcguinness.html](http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html). Date Accessed: March 25, 08.
17. Ravi Sandhu, David Ferraiolo, Richard Kuhn, “The NIST Model for Role Based Access Control: Towards a Unified Standard”, URL: http://csrc.nist.gov/rbac/sandhu-ferraiolo-kuhn-00.pdf.
18. Niklas Borselius, Mobile VCE research Group, Royal Holloway, University of Lundon, “Mobile Agent Security.”
19. Levent Ertaul, Jayalalitha Panda, “Mobile Agent Security.”, URL: http://iec.cugb.edu.cn/WorldComp2006/SAM3870.pdf. Date Accessed: March 25, 08.
20. Stephen R. Tate, Ke Xu, “Mobile Agent security Through Multi-Agent Cryptographic Protocols”, URL: http://cops.csci.unt.edu/publications/2003-01/2003-01.pdf. Date Accessed: March 25, 08.
21. Carl Ellison, Steve Dohrmann, “Public Key Support for Group Collaboration.”, URL: http://portal.acm.org/citation.cfm?id=950195.
22. Chih-Lin Hu, Wen-Shyen E. Chen, “Mobile Agents Collaboration for Information Gathering.”

(THE END)

1. An interface that is basically a wrapper to hold all Content Facilitators. This is the super interface for all Content Facilitators. [↑](#footnote-ref-2)
2. Extends CFObject and contains variables to store ACL content parameter. [↑](#footnote-ref-3)
3. The final application to design that will demonstrate the agents’ communication scenario. It will be described later. [↑](#footnote-ref-4)
4. Next section will describe how an ACL message will be enveloped and de-enveloped into PKCS 7. [↑](#footnote-ref-5)
5. Agent that will play the role of a post man and deliver the message to the attended recipient. [↑](#footnote-ref-6)
6. DistinguishedName object represent globally unique name. Since PKCS 7 relies on Public Key Infrastructure, therefore these objects are included to invoke its certain functions. The detailed description is beyond the scope of this work. Similar is the case with ASN1Encoding data item. [↑](#footnote-ref-7)