APPLICATION GROUP SUPPORT INFRASTRUCTURE FOR OCTOPUS: A MULTIMEDIA COMMUNICATION MIDDLEWARE

XIAO DONG-CHEN
(*B. Eng., Shanghai Jiaotong University*)

A THESIS SUBMITTED
FOR THE DEGREE OF MASTER OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE
2003

Acknowledgements

My foremost acknowledge goes to my research supervisor, Associate Professor Pung Hung Keng, for his invaluable directions and support throughout my research efforts towards this thesis. His insights and suggestions to the problems in this thesis enlightened me in various detailed aspects throughout the work.

My acknowledge goes to my OCTOPUS team members. Among them, my special thanks go to Chaiwat Siriyuenyong, Robin, An Liming and He Jun, who have been generously spending their precious time discussing with me research issues related with my work, as well as to helping me in proofreading this thesis.

Last but not least, my family deserves particular recognition for their unconditional emotional support during the past years, even though we have been far away from each other. I'm greatly in debt to my parents who have brought me up with so much love and care throughout the years.

Table of Contents

CHAPTER 1 INTRODUCTION....1

1.1 A GROUP APPLICATION SCENARIO....1

1.2 REQUIREMENTS....3

1.3 MOTIVATION....4

1.4 CONTRIBUTIONS....7

1.5 THESIS ORGANIZATION....9

CHAPTER 2 BACKGROUND AND PROPOSED FRAMEWORK....10

2.1 RELATED WORK....10

2.1.1 MBone related session protocols....10

2.1.2 Group and Session Management....13

2.1.3 Computer-Supported Cooperative Work and Groupware....16

2.1.4 Peer-to-Peer Network Architecture....18

2.2 PROPOSED PREVIOUS OCTOPUS FRAMEWORK....20

2.2.1 Stream Service Architecture in OMG AV Spec....20

2.2.2 OCTOPUS Architecture....22

2.3 CONCLUSION....24

CHAPTER 3 AGSI ARCHITECTURE AND CORE DESIGN....26

3.1 AGSI ARCHITECTURE....26

3.1.1 Architecture Anatomy....28

3.1.2 Strength of AGSI....30

3.2 AGSI GROUP AND MEMBERSHIP MANAGEMENT....31

3.2.1 Groups in AGSI....31

3.2.2 Membership Management in AGSI....35

3.3 AGSI ACCESS CONTROL AND SECURITY....36

3.3.1 AGSI Access Control Model....36

3.3.2 Applying digital signature technology into AGSI....36

3.3.3 Conclusion....37

3.4 AGSI SESSION ORCHESTRATION....38

3.4.1 AGSI Session Sequence Diagram....38

CHAPTER 4 AGSI PROTOCOLS....41

4.1 AGSI MEMBERSHIP MANAGEMENT PROTOCOL....41

4.2 AGSI MEMBERSHIP ESTABLISHMENT PROTOCOL....43

4.2.1 Approval-based Membership Establishment....44

4.2.2 Open Group Membership Establishment *46*
4.3 AGSI DISCOVERY PROTOCOL 47
4.4 AGSI SEARCHING PROTOCOL 50
4.4.1 Search for Local Sessions *50*
4.4.2 Search for Remote Sessions *52*
4.5 AGSI GROUP-TO-GROUP SESSION BRIDGING PROTOCOL 55
4.5.1 Introduction of OCTOPUS network-level group bridging *55*
4.5.2 Bridging Two Session Groups at the Same AGSI Server *56*
4.5.3 Bridging Two Session Groups at Different AGSI Servers *58*
4.5.4 Partial Member Joining in Group-to-Group Bridging *59*
4.5.5 Cascading Group-to-Group Bridging *60*
CHAPTER 5 AGSI IMPLEMENTATION AND EVALUATION 63
5.1 AGSI SERVER DESIGN AND IMPLEMENTATION 63
5.1.1 AGSI Session Scheduler *64*
5.1.2 AGSI Session Manager *65*
5.1.3 AGSI Data Manager *66*
5.1.4 AGSI Security Manager *66*
5.1.5 AGSI Members Pool *67*
5.1.6 AGSI Sessions Pool *67*
5.1.7 AGSI Data Store *69*
5.1.8 AGSI Profile XML Configuration *69*
5.1.9 AGSI Server classes diagram *69*
5.1.10 AGSI Server Main Flow Diagram *70*
5.2 AGSI PEER DESIGN AND IMPLEMENTATION 71
5.2.1 AGSI Proxy *72*
5.2.2 AGSI Agent *72*
5.2.3 AGSI Session Configuration *72*
5.2.4 AGSI Profile XML Configuration *72*
5.2.5 AGSI Session Container *73*
5.2.6 AGSI Peer Heartbeat *73*
5.3 AGSI GROUP-2-GROUP BRIDGING/DISBANDING 75
5.4 AGSI TEST-BED 79
5.4.1 Test-bed Configuration for general session operation *79*
5.4.2 Test-bed configuration for AGSI Group-2-Group operation *80*
5.5 EVALUATIONS 84
5.5.1 AGSI System Bootstrapping and Session Management *85*
5.5.2 AGSI Session Publication *86*
5.5.3 Session Directory Retrieval *87*
5.5.4 Group-2-Group Operations *89*
5.6 DISCUSSION AND CONCLUSION 91

CHAPTER 6 CONCLUSION AND FUTURE WORK 92
6.1 CONCLUSION 92
6.2 FUTURE WORK 93
6.2.1 Provision of a pure P2P computing model 93
6.2.2 A Unified Identity Management System 94
6.2.3 Improvement in Security 94
6.2.4 Introducing Web Service Technology into AGSI 94
APPENDIX A CODE SNIPPETS IN INVOKING AGSI API 95
PART 1: AGSI SERVER INITIALIZATION 95
PART 2: AGSI PEER INITIALIZATION 96
PART 3: CREATION OF SESSION CONTAINER IN AGSI PEER 96
PART 4: CREATION OF MULTIMEDIA DEVICE IN AGSI PEER 96
PART 5: PUBLISHING OF MULTIMEDIA APPLICATIONS 97
PART 6: RETRIEVAL OF AGSI SESSIONS DIRECTORY 97
PART 7: SESSION INITIALIZATION AT BOTH SIDES 97
PART 8: BRIDGING OF SESSION GROUPS 97
APPENDIX B AGSI CONFIGURATION XML 98
PART1: AGSI SERVER CONFIGURATION XML 98
PART2: AGSI PEER CONFIGURATION XML 99
PART3: AGSI MULTIMEDIA DEVICE CONFIGURATION XML 100
REFERENCES 101

Summary

The Internet is used today not only as a global information resource, but also to support collaborative applications such as voice- and video-conferencing, distributed simulations, white boards, multi-party games and replicated servers of all types. Our research project OCTOPUS [1] was intended to provide middleware[1] supports to those upper level applications. OCTOPUS simplifies the setting up of real-time stream communication between two parties through the use of stream-APIs; its dynamic protocol framework allows protocol stacks of end-hosts (e.g., transport protocols and codec stacks) be configured dynamically for meeting end-to-end needs; its Connection Manager extends the semantic of managing multicast membership from managing discrete users as multicast members to managing groups as multicast members.

The existing OCTOPUS framework lacks the following features: (I) membership management and manipulation functions at high level (such as to applications) which are essential for supporting collaborative applications; (II) access control to application sessions, which is particularly important to OCTOPUS as its low-level connection management functions are open and hence do not enforce security and access control at that level; (III) common session control and management functions, such as the description, advertisement and initiation of a session and other intra- and inter-session support for group membership management. Consequently, application programmers of OCTOPUS have to figure out their own way of managing and binding end-users (devices, process or human users) to various sessions. To address this shortcoming, we

[1] Middleware: layer(s) of software between client and server processes that deliver the extra functionality. It hides the complexity of the extra functionality behind a common set of APIs that client and server processes can invoke.

proposed and implemented a new collaborative applications supporting framework known as application group support infrastructure (hereafter referred to as AGSI in short).

In an AGSI supported environment, every end-host or end-user is known as an *AGSI peer*. Each peer contains AGSI enabling components through which the peer can interact and be managed by the associated application group servers known as *AGSI servers*. These application group servers manage the group membership data and provide AAA (authentication, authorization and auditing) services to the application groups and their members. Furthermore, session orchestration is an important feature of AGSI. It facilitates the establishment or teardown of group-based sessions in OCTOPUS. A membership and session manager (a part of the AGSI server) together with its corresponding client components known as AGSI agents (resides in each AGSI peer) orchestrate and manage the corresponding sessions. Within each AGSI peer, there is an *OCTOPUS multimedia device component* and an *AGSI proxy component*. The OCTOPUS multimedia device component does the data transportation for the multiparty multimedia system. The AGSI proxy component provides basic functionalities for a peer like managing its profile (records a peer's properties) and constructing AGSI agents. The AGSI agent component provides various handy functionalities including the application spec composition and session configuration. The application spec tells what a session is like according to its various session properties and how a session is composed. For instance, an application specification should provide configuration information of the streams and flows. It shall also provide information of the session controller and session access control list. The AGSI agent also helps advertise the session information to its associated AGSI server by passing the application specification that carries the session information. AGSI peers

can look up and discover sessions from their associated AGSI servers and request to join them if desired. Sessions are initiated and controlled (start/stop) by their corresponding session managers. Services or applications provided within each session can be shared to members from other sessions through our session orchestration service.

Our previous OCTOPUS framework mainly focuses on providing an infrastructure for data transport management. In contrast, the AGSI framework focuses on providing a group and session management support to collaborative multimedia application systems. Therefore, AGSI has greatly extended the OCTOPUS framework by addressing the session-level issues and providing some generic application-level support to high-level application systems.

List of Tables

TABLE 2-1: MULTIMEDIA SESSION RELATED PROTOCOLS 11
TABLE 2-2: A SURVEY OF GROUPWARE APPLICATIONS 17
TABLE 3: P2P NETWORK ARCHITECTURE COMPARISON 19
TABLE 3-1: 3-PHASE AGSI SESSION OPERATIONS DESCRIPTION 40
TABLE 5-1: AGSI SYSTEM BOOTSTRAPPING AND SESSION MANAGEMENT EXPERIMENTATION 86
TABLE 5-2: AGSI G2G OPERATIONS EXPERIMENTATION 90

List of Figures

FIGURE 1.1-1: APPLICATION GROUPS INTERACTION SCENARIO 1
FIGURE 2.1-1: GMS ARCHITECTURE 15
FIGURE 2.2-1: THE OMG AV STREAM SERVICE ARCHITECTURE 20
FIGURE 2.2-2: OMG AV STREAM SERVICE COMPONENTS 21
FIGURE 2.2-3: PREVIOUS OCTOPUS ARCHITECTURE 22
FIGURE 3.1-1: AGSI-ENABLED OCTOPUS ARCHITECTURE 27
FIGURE 3.2-1: AGSI GROUPS 32
FIGURE 3.2-2: AGSI MANAGING SCOPES 34
FIGURE 3.3-1: ASYNCHRONOUS CRYPTOGRAPHY 37
FIGURE 3.4-1: AGSI 3-PHASE SEQUENCE DIAGRAM 39
FIGURE 4.1-1: AGSI ID FACTORY AND ITS NAMING CONVENTION 42
FIGURE 4.2-1: MEMBERSHIP APPROVAL PROCESS 45
FIGURE 4.3-1: SERVICE DISCOVERY IN AGSI 47
FIGURE 4.3-2: EXAMPLE OF AN AGSI GROUP ADVERTISEMENT 48
FIGURE 4.4-1: SEARCH LOCAL SESSIONS 51
FIGURE 4.4-2: AGSI SEARCH FOR REMOTE SESSIONS 53
FIGURE 4.5-1: SESSION GROUPS MERGING 56
FIGURE 4.5-2: AGSI SESSION GROUPS BRIDGING AT THE SAME AGSI SERVER 57
FIGURE 4.5-3: AGSI SESSION GROUPS BRIDGING BETWEEN TWO AGSI SERVERS 58
FIGURE 4.5-4: PARTIAL MEMBER JOINING 59
FIGURE 4.5-5: CASCADING GROUP BINDING 60
FIGURE 4.5-6: GROUP BRIDGING ACCESS CONTROL 62
FIGURE 5.1-1: AGSI SERVER DIAGRAM 64
FIGURE 5.1-2: AGSI SESSION CONTAINER DATA STRUCTURE 68
FIGURE 5.1-3: CLASSES DIAGRAM FOR AGSI SERVER MAJOR COMPONENTS 69
FIGURE 5.1-4: AGSI SERVER BOOTSTRAPPING FLOW DIAGRAM 70
FIGURE 5.2-1: AGSI PEER DIAGRAM 71
FIGURE 5.2-2: AGSI PEER HEARTBEAT THREAD FLOW DIAGRAM 73
FIGURE 5.3-1: AGSI GROUP-2-GROUP BRIDGING DIAGRAM 78
FIGURE 5.4-1: GENERAL TEST-BED DIAGRAM 80
FIGURE 5.4-2: AGSI GROUP-2-GROUP TEST-BED DIAGRAM 82
FIGURE 5.4-3: AGSI GROUP-2-GROUP BRIDGING USING EMULATED CMS 83
FIGURE 5.5-1: POINT-TO-POINT OPERATION LATENCY DIAGRAM 84
FIGURE 5.5-2: AGSI PUBLICATION LATENCY 87
FIGURE 5.5-3: AGSI SESSION DIRECTORY RETRIEVAL 88

CHAPTER 1
INTRODUCTION

This chapter presents a collaborative application group environment and the concepts developed for the application group supporting system. It also highlights the potential shortcomings of the existing OCTOPUS middleware developed for supporting collaborative applications.

1.1 A GROUP APPLICATION SCENARIO

The various concepts used in Application Group Support Infrastructure framework, like application groups, group members and group sessions and how applications can be shared are to be elaborated using an e-learning application scenario as shown in Figure 1.1-1:

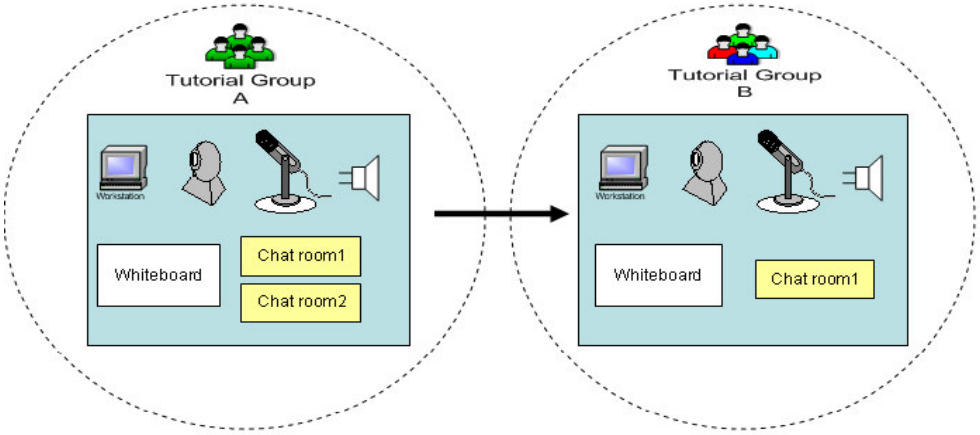

Figure 1.1-1: Application Groups Interaction Scenario

In the scenario, there are two instances of e-learning application group denoted as tutorial group A and tutorial group B, for the teaching of Java language and communication protocols respectively. Both application instances involved the uses of a video conferencing, a chat and a whiteboard to conduct the tutorials. The users engaging in Chat room 1 application from the tutorial group A can access to the chat room 1 application provided in tutorial group B, or tutorial group B is sharing one of its applications: chat room application to the tutorial group A.

Within each tutorial group, we treat the whole set of group activities as one big *application* which can be further divided into many *sub-applications*, like a whiteboard application, a chat room application and an audio/video streaming application as depicted in the diagram. In comparison, a run-time instance of an application is called a *session* for managing the run-time application states and the engaging users. A session may have multiple *sub-sessions* that are run-time instances of the sub-applications. All the users that can access to one application form the *application group* and those users are *application members* of that application group. Similarly, all the users that are engaging currently in one session form the *session group* and they are the *session members* of that session group. It is not difficult to see a session group is different from an application group since the session members may come and go in an uncertain manner from the network while application members can be defined at the application design time. To have a succinct view of the above concepts and their relationship, we describe them in following mathematical forms:

(1) Application and its sub-application:

$$Application_x = \{ App_{x1}, App_{x2}, ..., App_{xn} \}$$

(2) Session and its sub-sessions:

$$Session(Application_x) = \{Session(App_{x1}), Session(App_{x2}), ..., Session(App_{xn})\}$$

(3) Users of application and users of sub-application

$$Users(Application_x) =$$
$$\{user_i \mid \forall i.j, i \neq j, user_i \in Users(App_{xi}), user_j \in Users(App_{xj}), user_i \neq user_j\}$$

(4) Users of session and users of sub-sessions

$$Users(Session(Application_x)) =$$
$$\{user_i \mid \forall i.j, i \neq j, user_i \in Users(Session(App_{xi})), user_j \in Users(Session(App_{xj})), user_i \neq user_j\}$$

The above mathematical forms: (3) and (4) show that a user may access to more than one application and may also engage in more than one session, which all depends on the application and session access control settings.

1.2 REQUIREMENTS

Now we have established the concepts of application, application group, session and session group as well as application members and session members. These concepts stand for specific application entities and provide means and managing scopes for application programmers to conduct the membership management, access control and session management at an application- or session-level. It is very important to keep a meta- database to record these entities information. Furthermore, there should be a layer of managing service to retrieve and save the information from and to the meta-database. When the environment changes, the service shall be also responsible for updating the meta database and notifying all the engaging members about the change if necessary. Lastly the service shall provide means for different group members to interact with others.

1.3 MOTIVATION

As we understand the requirements provided in the above application scenario, let us look at a more general case of such application environment by learning its background and the possible requirements for establishing a collaborative group support system.

The advent of network and computer technologies has spurted the development of online services such as information dissemination, entertainment, education, e-business and e-commerce. They have enormous impacts on our daily life and in business. Millions of individual end users and thousands of global enterprises use computer networks as a platform for communication, collaboration and sharing of information routinely, as has been the case of using telephones for voice communication over telephone networks

One should note that the sessions established between service providers and users are run-time instances of these services or applications. Since both the presence of users and service providers may be dynamic, it becomes harder for individual entity (such as an user or a service provider) to gather the latest status of the parties involved Furthermore, the admission to group related services and applications is typically administrated and regulated by the respective administrative domain. Therefore, there is an urgent need of providing collaborative work support infrastructure for intra- and inter-group activities. Clearly the corresponding intra- and inter-group meta information has to be managed. For instances, the information of application group profile and the group membership as well as specific member profile has to be maintained somewhere, which in our case are our AGSI servers. Furthermore, the

group server shall be able to authenticate its members and direct them to the appropriate applications resources (i.e. through access control).

Session operations normally include session advertisement, session initiation and session control (specifying who can access a session and when to start/stop a session). Advanced session operations to provide collaborative support between different sessions, such as to allow all the engaging members of one session to access another one shall be included. All these session-related activities belong to the session management.

For the network architecture of the AGSI, we adopt a mixed mode between the client/server computing-model to a purely distributed one. We believe a centralized solution for group and session management can bring many benefits like efficient management of group sessions and high security for the session access for it is easier to realize interoperations and collaborations among various sessions that are managed by a single session manager. But to group the scope within a specific domain and not to mix the application group with other non-relevant ones, we assign multiple local application groups servers to manage and control these groups. Finally, these application group servers can communicate with each other through the common service discovery system provide in the present OCTOPUS framework.

Our research project OCTOPUS [1] represents a middleware support to application programmers by relieving them from writing low-level code like creating network-related components, multimedia-enabling components and devices/services discovery-enabling module. By adopting OCTOPUS middleware, application programmers can quickly set up a multimedia application environment equipped with some nice and

unique features like Stream Management Support, QoS[2] support and dynamic protocol adaptation framework [2][3][5]. Nevertheless, OCTOPUS lacks the important features that make it a collaboration-oriented middleware. Firstly, it does not support a group concept and thus fail to create group awareness for application users. Without a group support, it is very hard to create a secure environment for a user to enjoy various interesting group sessions. Secondly, it does not provide a unified session management support by standardizing how to describe a session, retrieve a session, initiate a session and control a session. It does not answer the question of where to reside the session meta information either. For other requirements such as interoperations between sessions, the present OCTOPUS has little to offer except application programmers have to figure out their own ways in implementing them. All these outstanding issues make the collaborative applications development based on the present OCTOPUS framework still a challenging task.

Therefore, in this thesis, a new architecture concept as termed as **Application Group Support Infrastructure (AGSI)** is proposed to extend the existing OCTOPUS framework for collaborative group applications. The AGSI framework aims at providing an infrastructural support, including group and session management, security and access control, to collaborative multimedia applications. With AGSI, many CSCW-related applications, i.e. groupware applications can be easily built and different groupware applications are thus able to collaborate among themselves [8][9][10].

[2] QoS: Quality of Service. Here we refer to network-related services QoS only.

1.4 CONTRIBUTIONS

AGSI establishes an application group and session management system that provides various services to the high-level applications. A powerful collaborative support system for multimedia group applications is created when AGSI is integrated with OCTOPUS. It simplifies the task of application programmers in the development of various collaboration-aware multimedia applications.

AGSI, intended as part of the OCTOPUS middleware, is embodied in both AGSI server and its clients or peers. Each of them can communicate to each other and provide service to each other if necessary. But all the AGSI peers connect to their associated AGSI servers for conducting sessions. Once a session is established between two AGSI peers, the AGSI server can be free from the engagement of that session. To conclude, AGSI has enhanced the services and functionality of OCTOPUS in three aspects:

- Group and Membership Management

  The group and membership information are kept and managed at AGSI servers for the corresponding application groups. The AGSI maintains information for different entities like application group profile, session group profile and application group users and session group users. AGSI membership service helps individual users to establish their membership with application groups and session groups. Furthermore, AGSI group manager runs at the AGSI server and plays as a broker for application users to look up, match and locate the right resources, services and applications offered within application groups.

- Access Control and Security Support

  AGSI has been designed to operate in environment where memberships and resources allocations are dynamic. This makes it very hard to pre-determine any specific role to some members. Therefore, It is becomes natural for us to consider using the Identity-based [24] or user-based Authorization Model, i.e. every resource will be associated with an access control list since we could hardly fix a role for individual resources to manage the access control. But in some cases, we can conceive there could be some common roles among an application group and members with the roles shall be able to access some or all resources provided within that group. By binding the identity-based and role-based authorization methods together, we create a hybrid access control model, which produces great flexibility for the application system. This practice has been widely adopted in many contemporary file systems, e.g. Microsoft Windows NTFS. Besides granting access control at the application level, we also provide functional level security support by enforcing the authentication during each method call to make secure communication.

- Session Orchestration Service

  AGSI session managers manage various sessions in a higher level for the session group members. It maintains a session pool for session producers to advertise and for session consumers to discover and download session information for their consumption. It is also responsible for initiating and establishing the sessions by binding the endpoints of individual session flows after receiving the session-joining request from every session party. Beside these intra-session operations, AGSI session managers also help share

applications from one session group to another, meaning that an application provided within one session group can be immediately shared to another session group.

The above three building blocks lay the foundation for most multimedia-intensive, security sensitive and collaborative systems. Due to its layered and modularized structure design, one can easily extend the AGSI to another level to provide more application-related services.

1.5 THESIS ORGANIZATION

Chapter 1 is the introduction to this whole thesis. Chapter 2 further introduces the background of this research and previous work of OCTOPUS middleware. Following that, Chapter3 gives an overview of the AGSI architecture and elaborates the main components that make up the AGSI framework. Most of the terminologies used throughout the thesis are also defined in this chapter. Chapter 4 presents the design of the protocols under the AGSI framework. Chapter 5 describes the implementation of a prototype AGSI system in and presents the results of a performance study. We conclude the thesis and highlight possible future work in Chapter 6.

CHAPTER 2
BACKGROUND AND PROPOSED FRAMEWORK

In this chapter, we present an extensive survey we made on various research areas that are closely related to our research work. We also give an introduction of the OCTOPUS framework upon which our AGSI framework is built.

2.1 RELATED WORK

With more and more computers having built-in multimedia capability and becoming networked workstations, many research areas like multipoint multimedia communications, group and session management, CSCW and groupware, have received much attention recently. Closely related to these areas, AGSI aims to provide an infrastructure-level support to multimedia and collaborative applications that requires high-level group and session management.

2.1.1 MBone related session protocols

MBone, short for Multicast Backbone [16], is a virtual network that provides one-to-many and many-to-many network delivery services for applications such as videoconferencing and audio where several hosts need to communicate simultaneously. Video, audio, and a shared drawing whiteboard are the principal MBone applications. Another multicast application called *sd* (session directory), which displays active multicast session groups, was developed by Steve McCanne and Van

Jacobson of the University of California Lawrence Berkeley Laboratory. The *sd* tool is based on the Session Description Protocol (SDP) v2 and the Session Announcement Protocol (SAP) described by Handley and Jacobson [17][18]. However, since SDP can only be used for session advertisement and SAP for the distribution of announcements, an additional protocol is required which is used for specifically inviting users to sessions. This protocol is the Session Initiation Protocol and is described by Handley et al [19]. A more detailed description of the above three protocols and their relevance to AGSI framework are listed in following Table 2-1.

Protocol	Description	Relevance to AGSI framework
SIP: Session Initiation Protocol	It directly targets every individual addressable online users rather than groups. It specifies how to initiate an interactive user session that involves multimedia elements such as video, voice, chat, gaming, and virtual reality	A group-based session initiation support provided in AGSI Session Manager
SDP: Session Description Protocol	SDP is a protocol that defines a format for conveying descriptive information about multimedia sessions. When a user wants to join a conference, he or she needs a way to know the multicast group address and the UDP port address for the conference. SDP was designed as a session directory tool that could be used to advertise multimedia conferences, and communicate the conference addresses and conference tool-specific information necessary for participation. At the same time, SDP was designed for general-purpose use so that it could be useful for a wide range of network applications	An AGSI application specification can describe sessions and session groups that are run time instances of applications and a group of applications within a collaborative environment.
SAP: Session Advertisement Protocol	A protocol to announce Internet multicast conferencing sessions. A conference is announced by periodically multicasting a UDP announcement packet to a multicast address and port. Because SAP is designed for multicast, it is suitable for setting up conference calls, not one-on-one IP telephone calls.	AGSI Session Manager, as a central place, provides an information repository to members to publish sessions information and allows authorized users to retrieve the sessions advertisement

Table 2-1: Multimedia Session Related Protocols

As shown from the above table, our AGSI framework realizes the functionalities provided by the three protocols but in different ways:

(1) For session advertisement, we let AGSI session manager realize this functionality without relying on an open MBone virtual network since we deem the application group server where AGSI session manager resides in is a better place for hosting such information in terms of providing a more controlled group environment and avoiding unwanted users from accessing it.

(2) For session description, we adopt the basic ideas of how to describe a multimedia session and extend it to describing multimedia applications that may contain multiple sessions and other session related properties like QoS requirements.

(3) For session initiation, AGSI supports inter- and intra- group application. SIP is based on a HTTP-like request/response and its session management model is initiator-based[3] [11]. In contrast, the session initiation management in AGSI is based on remote object invocation method and is joiner-based[4]. Choice of remote object invocation makes the session initiation design tightly coupled with the implementation details, which makes the session management less extensible. However, it offers two advantages that SIP does not enjoy: one is the achievement of runtime performance due to the binary data transmission rather than the HTTP-like textual information transmission; another is easy in

[3] Initiator-based: Through some sequence of dialogs the initiating user invites other users to the collaborative session. The number of invitations issued can be potentially large, depending on the application and the context of the task. Invited users can accept or reject the invitation.

[4] Joiner-based: The initiating user creates a new session; user must find the session by browsing the list of currently active session (or know a priori that the session will be taking place). Once they know the session handle they can attempt to join the session.

management and maintenance of codes since remote object invocation is more object-structured and easier for code management. Furthermore, SIP adopts a connection-less communication model while AGSI adopts the connection-based communication model, both of which have their pros and cons. In contrast, SIP transactions require the responders parse the textual information before it can execute the commands conveyed in every SIP message. Lastly SIP targets individual users and lacks a group-level support which is required by collaborative communication systems. On the contrary, AGSI session initiation does not suffer such a problem due to its application-level group support that can keep session initiation happen only at certain groups rather than at a worldwide open environment.

2.1.2 Group and Session Management

Distributed multimedia applications require group and session management from the underlying group communication platform. The group management deals with the dynamicity of users and groups of users and their relationships with applications, while the session manages and controls the establishment of communication between these users and the applications. The followings survey the research works in the areas of group and session management.

(1) Group Management Service (GMS) [13] is designed to support collaborative interactions among groups of distributed users with different applications. The model of the GMS is very simple and consists mainly of two classes of objects, namely user and group. A small set of operations is provided for querying and modifying GMS information. The same idea has been introduced to AGSI framework for abstracting collaborative applications from a common database

of information about users and groups. But AGSI extends the group concept to the session level where all involved session parties spontaneously form a dynamic session group and that information shall be kept in AGSI framework in order to provide group-awareness among session group members.

(2) Session Management Service (SMS) [11][12] coordinates all the necessary operations from starting a session until its end. It acts as a mediator between the users and the involved services by providing a set of operations that are grouped into core session management (e.g. open/close the session), participant management (e.g. invite a participant) and floor control (e.g. assign/revoke the floor) and application management (start/terminate a service). There are initiator-based and joiner-based session management models, both of which belong to *explicit session management* since the participants in the collaboration are required to take some action (perhaps time consuming) to join the session. However, more spontaneous collaboration fit better into a model called *implicit session management,* which avoids the overhead of the explicit session creation, naming and browsing phase. Examples of such model are serendipitous meetings in a hallway or in a break room. In AGSI, session management belongs to the explicit model and is joiner-based. We choose this model is because we target those application environment that requires a high degree of formality or where there is a natural name for the activity. An analogous "real-world" situation is "Java tutorial on Wednesday at 2:00 PM in room 302 for undergraduate students of class 2003-1." This task is widely known to the participants, is at a well-know location, and embodies a degree of formality.

(3) Group and Session Management (GMS) architecture [12] extended the previous work [11] by the same author Erik Wilde, et al, at Swiss Federal Institute of Technology in 1996. This model combines group management and session management together for an environment that requires multipoint multimedia group communications and collaboration. The GMS architecture, as shown in Figure 2.1-1, supports multipoint multimedia data transport services and separates the whole system into different planes, one dealing with the actual data transfer and another being responsible for management issues. The main architectural components of GMS are GMS user agents (GUA) and GMS system agents (GSA). While GUAs are included in the group communication frameworks using GMS, GSAs are stand-alone components, which, in their entirety, make up the GMS database that stores the relatively permanent information about users and user groups.

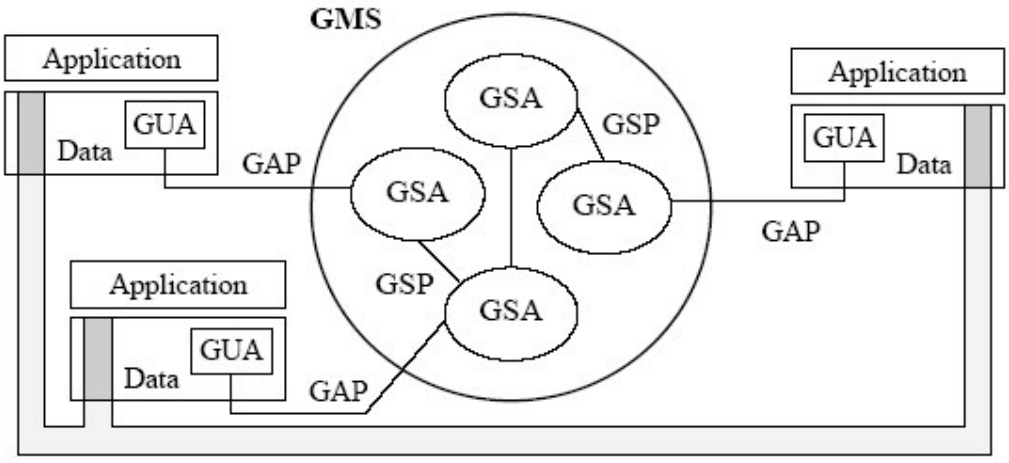

Figure 2.1-1: GMS architecture

In the AGSI framework, we adopt an analogous architecture by having the AGSI Agents as the GUAs and AGSI servers as the GSAs, except that the relationship among AGSI servers is more loosely linked through the SLM system [4]. Furthermore, the AGSI framework is directly built on top of the

OCTOPUS framework thus achieve a transport-independent Group and Session Management for group communication platforms [14]. The details of AGSI framework will be discussed in next chapter.

2.1.3 Computer-Supported Cooperative Work and Groupware

One of the main emphases of the chair of Applied Informatics-Distributed Systems is on computer support for teamwork. Activities from that domain are known by the notions of groupware or by that of computer-supported cooperative work (CSCW). Ellis defines groupware as "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment."[8] Typical topics include use of email, hypertext that includes awareness of the activities of other users, videoconferencing, chat systems, and real-time shared applications, such as collaborative writing or drawing.

Key issues of CSCW are group awareness[5], multi-user interfaces, concurrency control, communication and coordination within the group, shared information space and the support of a heterogeneous, open environment which integrates existing single-user applications. CSCW systems are often categorized according to the time/location matrix using the distinction between same time (synchronous) and different times (asynchronous), and between same place (face-to-face) and different places (distributed). The main purpose of CSCW is to facilitate group communication and productivity.

The definition for Groupware is “Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared

[5] Awareness: it refers to the capacity of participants within a group activity to perceive the actions, capabilities, and availability of others.

environment" (Ellis). Groupware is often used to specifically denote the technology that people use to work together, whereas CSCW refers to the field that studies the use of that technology. Examples of existing groupware applications and their features lists can be seen from below Table 2-2:

Groupware Application	Main Features List						Vendor
	PowerPoint	White board	Chat room/ Brainstorm	Application Sharing	Work flow	Multimedia Support	
Placeware	√	√	√	√	X	X	Microsoft
Teamwave	√	√	√	X	X	X	Teamwave
Netmeeting/ Outlook	√	√	√	X	√	√	Microsoft
Lotus Notes	X	X	X	X	√	X	IBM
Octopus+ AGSI-based E-Learning	√	√	√	√	X	√	SoC. NUS

Table 2-2: A survey of groupware applications

OCTOPUS, as a multimedia communication middleware, does not provide any support to address CSCW issues, even though the OCTOPUS middleware was originally introduced for enabling multiparty and multimedia application systems. Therefore, it would be a significant contribution to have a new layer that would sit between the OCTOPUS layer and the application layer and can provide the important features advocated by CSCW. In other words, the AGSI framework provides a common space for different application group users to store, retrieve and update information required for substantializing collaboration. It also provides a unified way of managing various application sessions and allows interactions between group members and application sessions.

2.1.4 Peer-to-Peer Network Architecture

Taking a distributed computing model, Peer-to-peer (P2P) offers unique set of benefits for dealing with unchecked growth in the number of connected users and devices, content, bandwidth, applications, and computing power. True peer-to-peer computing makes it easier and more intuitive for users to find and share resources. A peer-to-peer application is different from the traditional client/server model because involved applications act as both clients and servers. That is to say, while they are able to request information from other servers, they also have the ability to act as a server and respond to requests for information from other clients at the same time. A typical peer-to-peer application has the following key features that help define it:

1) Discovering other peers

2) Querying peers for content

3) Sharing content with other peers

There are a number of design options to consider. The range of applications in this area can be thought of as a continuum from what pure peer-to-peer to client/server. Table 3 shows four types of network architectural model for peer-to-peer communication that ranges from a pure Peer-to-Peer model to a Client/Server model. AGSI adopts a model that falls in one of the four models. In the AGSI system, peers can locate each other and discover interested sessions by querying from these local centralized AGSI servers. However, an AGSI peer (not the AGSI server) does not need to keep information of other AGSI peers. Furthermore, AGSI peers may also directly communicate with other peers for conducting any application sessions.

Network Architecture Model		Feature 1	Feature 2	Feature 3
		Discovering Other Peers	Querying Peers for Content	Sharing Content/Application with Other Peers
1	Pure P2P	Via peers	Via peers	Via peers
2	P2P with Peer-Discovery Server	Via centralized Peer-Indexing-Server	Via peers	Via peers
3	P2P with Peer-Discovery Server & Content-Discovery Server	Via centralized Peer-Indexing-Server	Via centralized Content-Indexing-Server	Via peers
4	Client/Server	Via server	Via server	Via server

Table 3: P2P Network Architecture Comparison

As discussed above, AGSI architecture model falls within the third category, i.e. the P2P model with peer-discovery server and content-discovery server. Because of individual AGSI servers manage local peer groups and distribute the workload among the groups in a hierarchical fashion, this model is expected to have better scalability and performance. The AGSI server offers the basic functionalities like peer indexing sand content indexing, which leads to fewer round-trips for peers in searching of other peers and contents. Compared to many existing pure or semi- peer-to-peer systems, the AGSI framework not only provides user and group management service but also provides session management service that is critical for application initiation and sharing among users and groups. Therefore, AGSI architecture gives stronger support for developing network-based multimedia applications.

2.2 PROPOSED PREVIOUS OCTOPUS FRAMEWORK

This section presents the OCTOPUS framework implemented in the Network Systems and Services lab. The prototype provides the necessary research and development environment for this thesis work.

2.2.1 Stream Service Architecture in OMG AV Spec

OCTOPUS offers a set of stream services to multimedia transmissions and adopts the architecture of the OMG AV specification for CORBA [7] as shown in Figure 2.2-1. It specifies how two end-points can be managed to communicate and transfer multimedia data in an efficient way through separating the data channel from the control channel. A data channel is responsible for transmitting multimedia data like audio or video stream data, whereas a control channel is responsible for sending controlling message like starting/stopping a streaming. The underlying network communication is built upon ORB Core of CORBA.

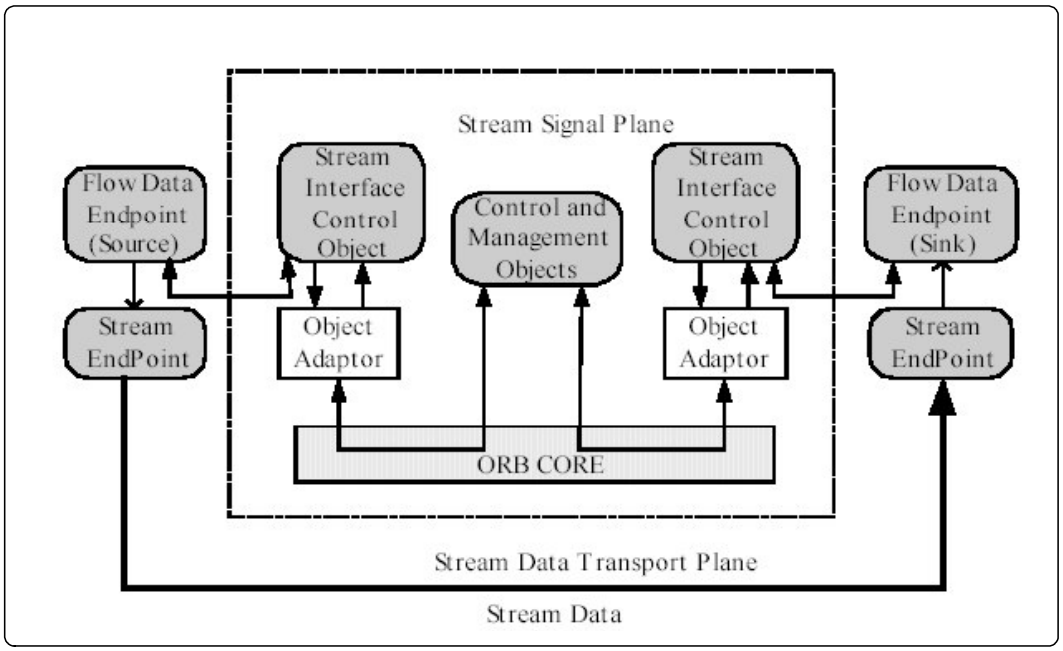

Figure 2.2-1: The OMG AV Stream Service Architecture

To have a closer view of how the components are composed and connected to complete the streaming service, let us look at another diagram as shown in Figure 2.2-2. It shows that there are three components that closely work together, namely stream controller and sender multimedia endpoint and receiver multimedia endpoint. In the stream endpoint, a multimedia device can manage multiple stream endpoints, each of which is further composed by multiple flow endpoints. Each data channel is in the form of a flow connection and is directly controlled by the stream controller that has the flow connection reference to the flow endpoints.

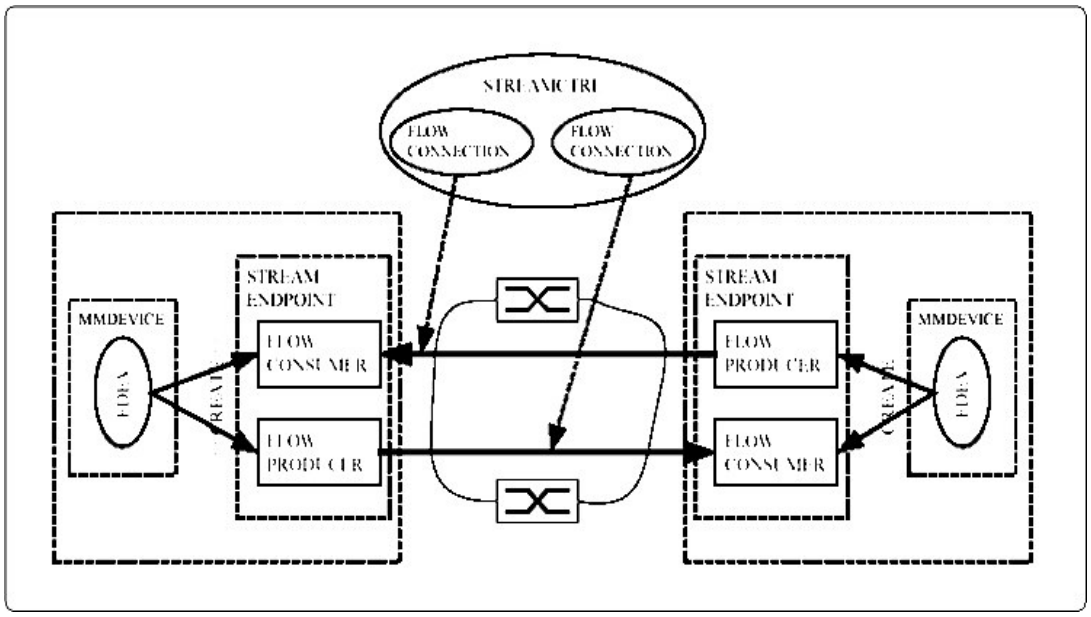

Figure 2.2-2: OMG AV Stream Service Components

The OMG AV stream service architecture and its detailed components design are very efficient and extensible. The AGSI framework is built upon the existing OCTOPUS framework that adapts CORBA's style of stream service architecture. In AGSI, there are application group servers that can host the stream control components for various streaming application peers. To avoid overloading of a group server, we also allow the

group server keep only the reference information of those streams; the latter may be created and hosted in physical locations other than the group server.

2.2.2 OCTOPUS Architecture

The architecture of OCTOPUS is evolving as the development effort of the OCTOPUS project team is continuing through the years. Figure 2.2-3 shows the OCTOPUS' architecture prior to the incorporation of AGSI.

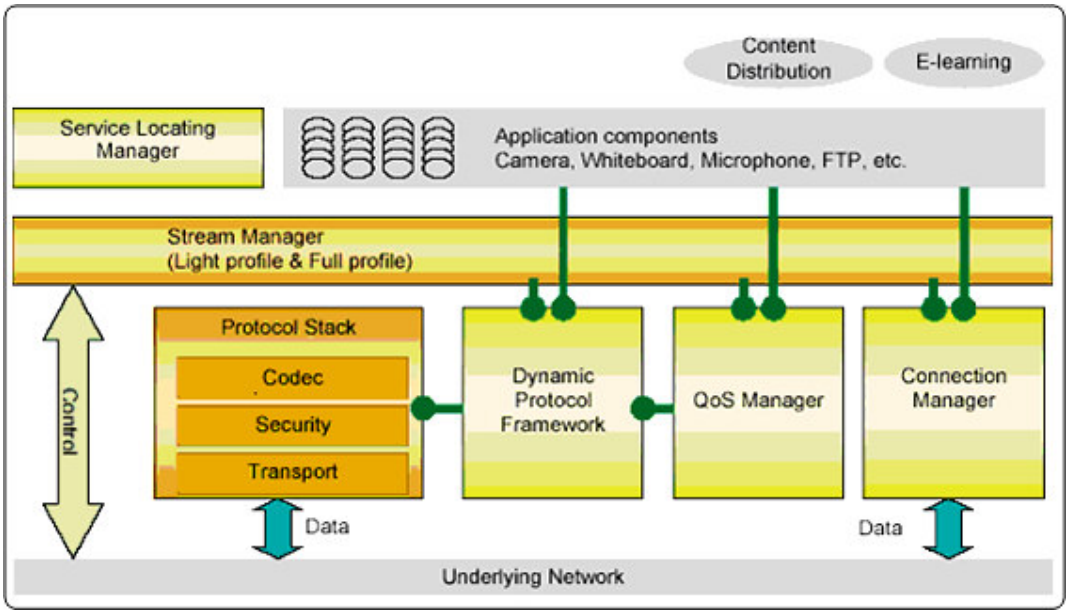

Figure 2.2-3: Previous OCTOPUS Architecture

From the architecture diagram, we note that the OCTOPUS middleware mainly duel with the group communications support, host-to-host quality of service support, and other related core middleware services. The features OCTOPUS of can be summarized as follow:

- It provides some useful and important network connection functions like setting up network sockets for point-to-point and point-to-multipoint multimedia streaming, enabling multicast to span multiple disconnected physical networks and enabling multicast group merging and disbanding.

- It offers two communication-enhancing features, namely dynamic protocol framework (DPF) and Quality of Service (QoS). DPF is created to allow dynamic switching of host's protocol stacks, from the transport stack to the presentation layer stack (such as media codec stacks). In doing so, DPF enables QoS adaptation to satisfy various customer needs and the changing network environment. The current implementation of QoS delivers a fairly scalable and effective QoS support for OCTOPUS applications.

- It provides a core middleware service known as Service Discovery service. There are already two versions of such service discovery service developed in OCTOPUS: one is through the use of SUN JINI [23] and the other is realized through a hierarchical Service Locating Manager (SLM) system and JINI [4].

Though these features are useful, the support for collaborative applications in existing OCTOPUS middleware remains weak. It lacks the concept of application groups and membership and thus lacks the features that are desirable in most collaborative systems. It also lacks a good support in the session layer that is critical to collaborative multimedia applications. Finally, the existing OCTOPUS provides a limited security support implemented in the form of encryption/decryption in the DPF component for data transmission. There is no security support for the use of OCTOPUS control channels, such as invocations of remote functional calls to do some infrastructural-level operations. Furthermore, there is no access control to the provided resources like services and applications or any other types of resources. It is preferable to handle the access control issues at the session layer.

2.3 CONCLUSION

Having reviewed the related works and status of OCTOPUS middleware, we have concluded that it would be neither sufficient nor convenient to build a multimedia-intensive communication and collaboration system simply with the support of previous OCTOPUS middleware framework. Application developers would still have to think about their own ways of implementing group-level communications, application sharing and collaboration, group and session management. If there were a new support layer which would standardize all these group-related collaborative operations and work flows, it would be a big leap forward for the OCTOPUS architecture in terms of solving mostly common real-life problems and achieving great efficiency to the industry.

AGSI, as a new initiative of enabling group-based communication and collaboration, is proposed to further empower OCTOPUS middleware and thus will further reduce the complexity of developing a powerful multimedia and collaborative system. Leveraging on all existing components of OCTOPUS, AGSI introduces many component-based modules at a higher level and make the whole architecture more complete and powerful for hosting collaborative multimedia applications. As a consequence of adopting of object oriented design and the choice of Java technology that is cross-platform, the AGSI architecture can achieve a higher degree of extensibility and manageability.

Finally, the computing model for the AGSI-enabled application system will be a hybrid of a Client/Server model and a pure P2P[6] model. There will be multiple AGSI servers for hosting application groups and there will be also many AGSI peers that

[6] P2P: peer-to-peer is about a communication model that is usually structured as one-to-one through an exchange system.

have OCTOPUS multimedia and communication capabilities within and can communicate directly with each other after obtaining the knowledge of through the AGSI group servers.

CHAPTER 3
AGSI ARCHITECTURE AND CORE COMPONENTS DESIGN

This chapter presents the core design ideas of AGSI. An architecture overview of AGSI is presented first. After that, the three areas of major contributions of AGSI from its functional perspective are explained in details. They are: the AGSI group and membership management, AGSI access control and security and AGSI session orchestration.

3.1 AGSI ARCHITECTURE

The AGSI architecture to be integrated with existing OCTOPUS components is depicted in Figure 3.1-1. A typical collaborative multimedia application system built with the OCTOPUS middleware consists of the following three parts from top to bottom, namely:

1) Applications
2) AGSI-enabled OCTOPUS middleware
3) Networks, which include the physical link, data link layer and the transport layers

Three sub-parts further compose the AGSI-enabled OCTOPUS middleware layer, namely: OCTOPUS, AGSI core and AGSI services. The AGSI core and the AGSI are

built on the existing OCTOPUS middleware and thus can be treated as an extension of the OCTOPUS middleware.

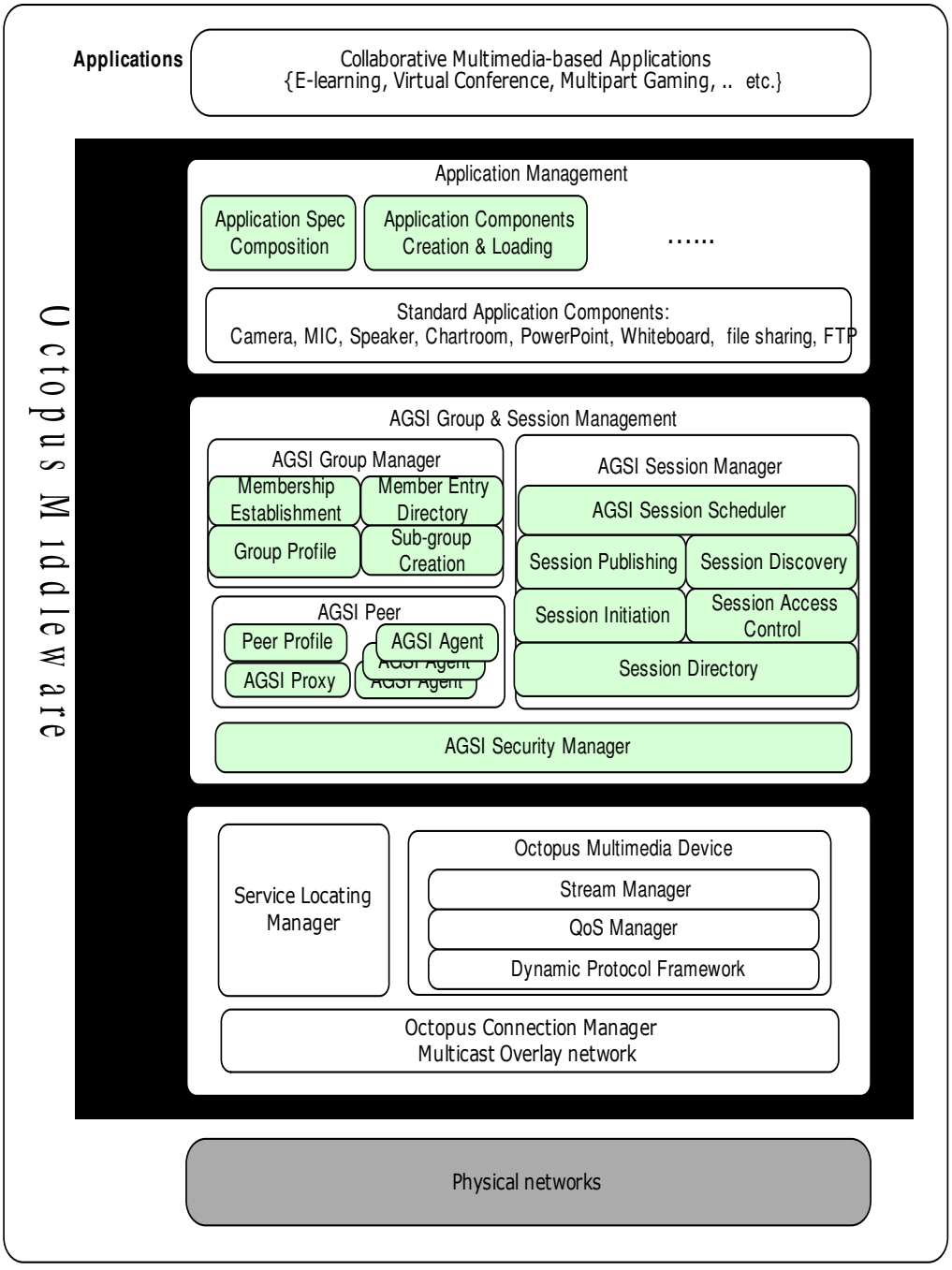

Figure 3.1-1: AGSI-enabled OCTOPUS Architecture

3.1.1 Architecture Anatomy

As shown from the above AGSI architecture diagram, the AGSI-enabled OCTOPUS middleware consists of the three main components: OCTOPUS, AGSI core and AGSI services. Here we analyze these components one by one, starting from the bottom up:

- At the very bottom of the OCTOPUS part is the OCTOPUS's Connection Management (CM) that enables conventional multicast service and group-cast operations (such as multicast group merging and disbanding). The CM is also responsible for the provision of multicast overlay [20] to bridge those separated multicast networks or IP multicast islands. Furthermore, CM is in a good position to enable the communication between any two end-hosts that are behind either NAT[7] servers or firewalls.
- On top of the CM is the OCTOPUS Service Locating Manager (SLM) [4] and the OCTOPUS multimedia device. SLM is a public information repository whereby any peer or service provider can register and publish some service information with it or simply look up for some service information. AGSI systems are relying on SLM to post and look up application group information. The OCTOPUS multimedia device is further composed by three key components: Stream Manager [2], QoS Manager [3] and Dynamic Protocol Framework [5]. They provide audio streaming, video streaming and data transmission support to higher level components.
- On top of the OCTOPUS is the AGSI core that handles AGSI group and session management. We group all the sub-components of this layer according to their functional roles. Physically these sub-components may reside at different network hosts, which in our implementation are categorized into two

[7] NAT: Network Address Translation, is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network.

types: the AGSI servers and the AGSI peers (application end-users that run AGSI client programs). In AGSI core, the four sub-components are:

1) The AGSI Security Manager, which resides at the AGSI Server and provides security support to AGSI session control channels by enforcing authentication and data encryption on every group and session management operation;
2) The AGSI Group Manager, which resides at the AGSI server and provides the group management operations like membership establishment, member entry directory, group profile management and sub-group creation;
3) The AGSI Session Manager, which resides at the AGSI server and does all the session-related jobs like: session publishing, session discovery, session initiation and session access control as well as maintaining the session directory that records all the ongoing sessions;
4) The AGSI Peer, which represents the individual application end-user, manages the profile data and the AGSI Proxy. Within the AGSI Proxy, there are as many AGSI Agents as corresponding application group servers, i.e. the AGSI servers. In later sections, all of these sub-components are to be further discussed in details.

- AGSI Services component mainly deals with the application management and is fairly extensible to multifarious collaborative application requirements. In this component, there is a module called Application Specification Composition that can capture most of the complex application requirements and put them into the standard application specification format. For example, if

one application is to be composed by one audio streaming, one video streaming and one chat-room, the Application Specification can store the requirements in XML format by adding the XML tags with the application component names: “<audio>”, “<video>” and “<chat-room>” respectively. With that Application Specification, through the Application Components Creation and Loading module, application programmers can quickly compose and set up various applications without spending time in implementing the applications themselves. Therefore, to maintain and extend the Standard Application Components module is very useful and can significantly save the development time for application programmers.

Having gone through the layers of AGSI-enabled OCTOPUS architecture, we will further look at what kind of outcomes AGSI brings about at the following section.

3.1.2 Strength of AGSI

Besides providing these basic functionalities for multiparty multimedia-oriented collaborative systems, AGSI also strives to achieve scalability, extensibility, ease of development and security for the whole system from the infrastructure level.

By scalability, AGSI can support any numbers of groups and members by assigning properly the members to groups in a proportional way such that every AGSI server can well handle the requests from its group members and can as well process the requests from other AGSI servers for group-to-group collaboration.

By ease of development, AGSI encapsulates most of the low-level programming jobs that are below the session level and makes them transparent to the application programmers. These jobs include creation of multimedia devices at one peer’s side and

creation of OCTOPUS Stream Controller at its server side and conducting all session-related operations.

By extensibility, it means AGSI architecture design are quite modularized and many of them can be plug-n-play, meaning that it can enable a feature by adding one separate module into AGSI as it can be disabled by being remove from AGSI. Furthermore, a third party can develop other modules based on AGSI infrastructure and put them together to make an enhanced version of AGSI.

Lastly, by security, AGSI achieves this in session layer through some measures like data encryption, authentication and authorization. The applications can make use of these components to ensure a secure application environment.

3.2 AGSI GROUP AND MEMBERSHIP MANAGEMENT

Throughout the OCTOPUS design, the concepts of group and membership have been frequently used but in different context and protocol levels. From the perspective of application layer, AGSI framework introduces the concept of groups and membership to solve application-related issues.

3.2.1 Groups in AGSI

AGSI group (also referred to as AGSI application group) is a representation of a virtual community consisting of a set of users who have participated in some common sets of applications. AGSI group members are those who have established their membership with the AGSI group. As we can see from Figure 3.2-1, AGSI group is a logical concept and thus is not confined to any computing platform, physical networks or geographical locations. Through the supports of AGSI group, members can communicate and collaborate with each other for conducting some applications like

audio/video streaming of a virtual conference. For simplicity, one AGSI server directly represents one AGSI GROUP. One can also create sub-groups within one AGSI Group in order to gain a finer control of application access scopes.

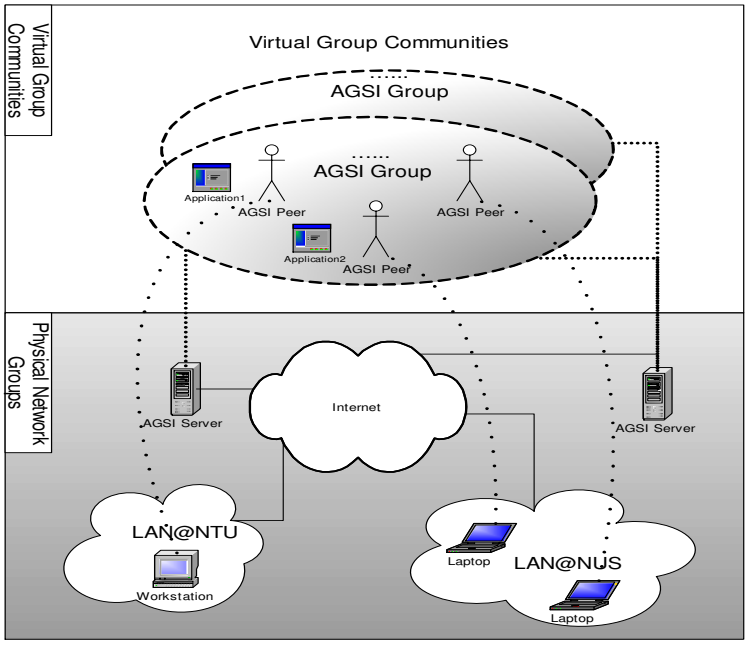

Figure 3.2-1: AGSI Groups

We recognize the following three motivations for creating AGSI application groups:

1. To create secure domains for exchanging secure contents or conducting secure services/applications. AGSI groups form logical regions whose boundaries limit access to non-members. An AGSI group does not necessarily reflect the underlying physical network such as those imposed by routers and firewalls. AGSI virtualizes the notion of routers and firewalls, subdividing the network in secure regions without respect to actual physical network boundaries.

2. To create a scoping environment. AGSI groups are typically formed and self-organized based upon the mutual interest of AGSI peers. No particular rules are imposed on the way AGSI groups are formed but peers with the same interest

will tend to join the same AGSI groups, of course after some possible authorization and membership establishment. AGSI groups serve to subdivide the network into abstract regions, providing an implicit scoping mechanism for restricting the effort of applications/services discovery.

3. To create a monitoring and governing environment. With the support of AGSI servers which are distributed as well, the activities of AGSI peers and the traffic, work load of the groups can be well managed and conducted for better efficiency and security.

We view the whole world as a single and virtual world group that may contain many AGSI groups that are represented and managed by their corresponding AGSI servers. Under each AGSI group or their subgroups, there can be various applications provided to its subscribed members or AGSI peers. The runtime instances of those applications are named as sessions and all the members engaging in the same session naturally form a session group that is dynamic and subject to changes determined by the engaging members. Figure 3.2-2 illustrates the scopes that AGSI system manages as described above.

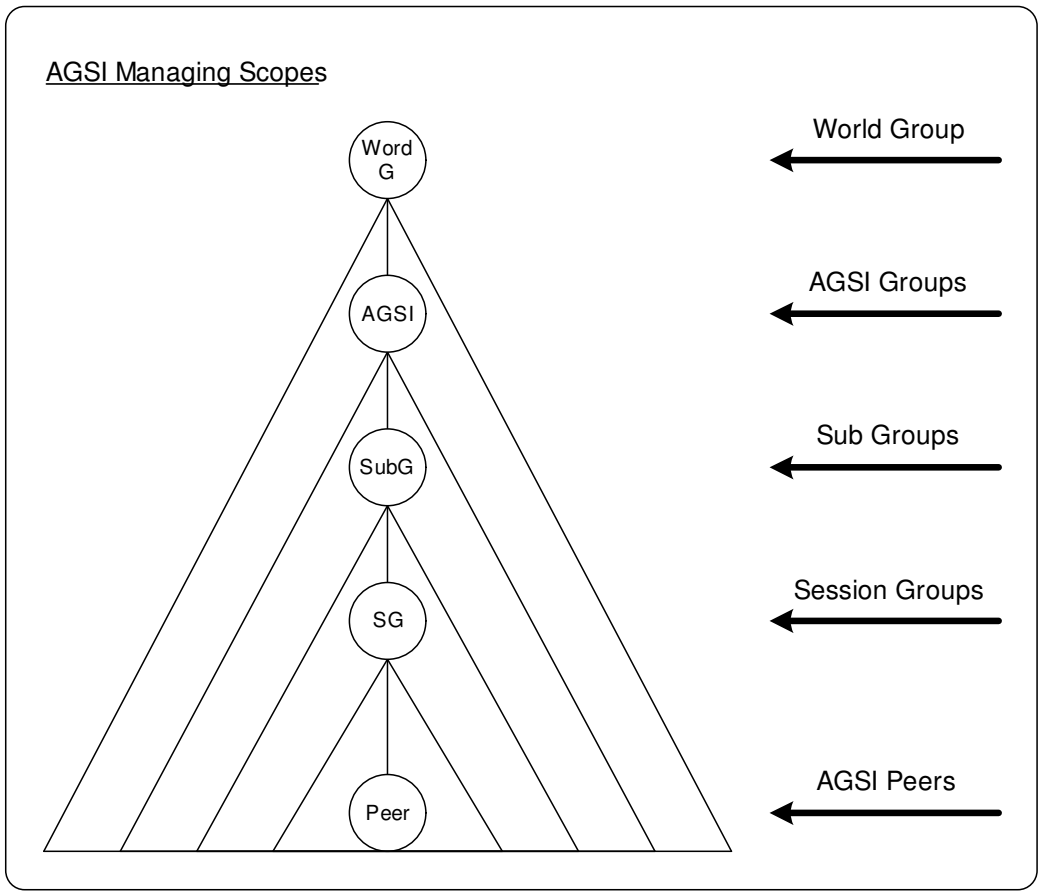

Figure 3.2-2: AGSI Managing Scopes

In our design, we allow the interaction happening at different levels like: AGSI-to-AGSI, SubG-to-SubG (SubG: a shorthand for Sub-groups), SG-to-SG (SG: a shorthand of Session Group) and even between the scopes that come from different levels. Each managing scope can be identified by a GUID[8] (except the world group, which is by default existing and one only) thus there will not be any problem in differentiating different groups from the same level or different levels. By the way, the session groups can be managed directly under the AGSI group or its subgroups.

[8] GUID: Global Unique Identifier, is a term used by Microsoft for a number that its program generates to create a unique identity for an entity such as a Word document (Equal to UUID: Universal Unique Identifier)

3.2.2 Membership Management in AGSI

AGSI membership management is about managing a set of AGSI Peers that belong to some AGSI groups. By assigning a set of AGSI peers to one AGSI group, it forms the scope of the AGSI group, which is not restricted by any physical factor like geographical location. Each AGSI peer is uniquely identified by its GUID.

Every AGSI peer has an XML-formatted peer-configuration file that records the basic profile information about the peer. The peer-configuration file also keeps the information of a peer's public-key and private-key that can be used for authentication upon initiating a session with some AGSI group. An example of such file can be found from the part 2 of the appendix B: AGSI Peer Configuration XML.

When an AGSI peer connects to the network, it will search for its subscribed AGSI groups and log onto any of them if it is desirable. The respondent AGSI server will create a member entry for this peer. Other members logging onto the same AGSI server will be aware of the new member and can retrieve the public information of that peer and even directly talk with it without the need to resort for any third party's assistance. But in most cases, it is still through the AGSI server to initiate and conduct AGSI sessions for one member with other members.

3.3 AGSI ACCESS CONTROL AND SECURITY

Security has always been important for most application systems. AGSI access control provides a session-level security control through its authentication and authorization. It also provides auditing of historical activities. Different members can have differentiated rights to access group resources, services or applications. In doing so, it helps protecting multifarious interests of members within a group.

3.3.1 AGSI Access Control Model

AGSI Access control is a hybrid of that of object-based model and role-based model. One member can authorize other individual member or a group or members with specific roles to access the applications, services or any resources provided by this member. One peer that is a non-member to that group or has no permission to some applications provided from that group is restricted from accessing those applications. However, a member can share his applications or any resources with others peers that are from other AGSI groups by granting them with the proper access rights. In conclusion, every resource and applications that needs access control shall be bundled with an access control specification that maintains a list of member entities, group entities or role entities with or without access right to that resource/application. Finally, the resource owner or the group administrator can make changes to the access control specification.

3.3.2 Applying digital signature technology into AGSI

The main issue in security is authentication or identification of a given object. In the proposed AGSI architecture, we will authenticate every remote functional call from a

specific peer with his digital signature that can be used to check if he is the one who owns the ID he claims to have.

Now let us have a brief look at the core part of digital signature technology that is being realized based on asynchronous cryptography, which is exemplified in Figure 3.3-1.

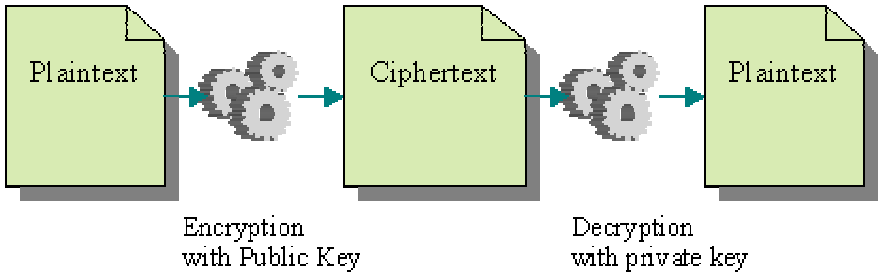

Figure 3.3-1: Asynchronous Cryptography

A peer will have his ID signed with his private key and present it to other party for authentication. The other party will check it with the peer's public key to see if he is the one who owns the ID and decide whether or not to continue the process as expected by the peer being authenticated.

3.3.3 Conclusion

AGSI access control and security support can help prevent unauthenticated peers from invoking any remote functional-level calls, which eliminates the odds of having AGSI security compromised. However AGSI access control and security support alone cannot fully guarantee the whole security of one real system that is based on AGSI and OCTOPUS middleware. The system application developers must make use of the AGSI security support and establish application-specific security practices together to ensure a holistic security protection for the system.

3.4 AGSI SESSION ORCHESTRATION

The key part of one AGSI server's job is to manage member sessions, which include session establishment, session advertisement, session retrieval, session initiation and all the rest of intra- and inter- session operations. On the other hand, AGSI peers, as the key elements of AGSI sessions, play an important part of these session operations. AGSI peers involve themselves in sessions with AGSI server and other peers to conduct various applications or collaborations within some applications. Another important role within AGSI architecture is Service Locating Manager or SLM. It helps individual peers to locate AGSI groups based on some criterions.

3.4.1 AGSI Session Sequence Diagram

The sequence diagram in Figure 3.4-1 depicts 3 phases of session operations involving AGSI servers, AGSI peers and SLM Servers. We assume the SLM servers have been configured and accessible from anywhere and anytime. The AGSI servers and AGSI peers may come and go frequently in a dynamic fashion. When the AGSI peers join the system, a list of AGSI servers will be downloaded from querying the SLM system. The AGSI peers can thus log onto different AGSI groups that are hosted and represented by the AGSI servers. Once again, a list of session information will be downloaded through querying AGSI servers. The session information contains the available applications that are provided by AGSI servers as well as the information on how to access these applications. The AGSI peers can decide to consume the application or start their own applications. During the midst of consuming a session, one consumer peer may leave the system and the AGSI session provider peer will be notified immediately through the AGSI server.

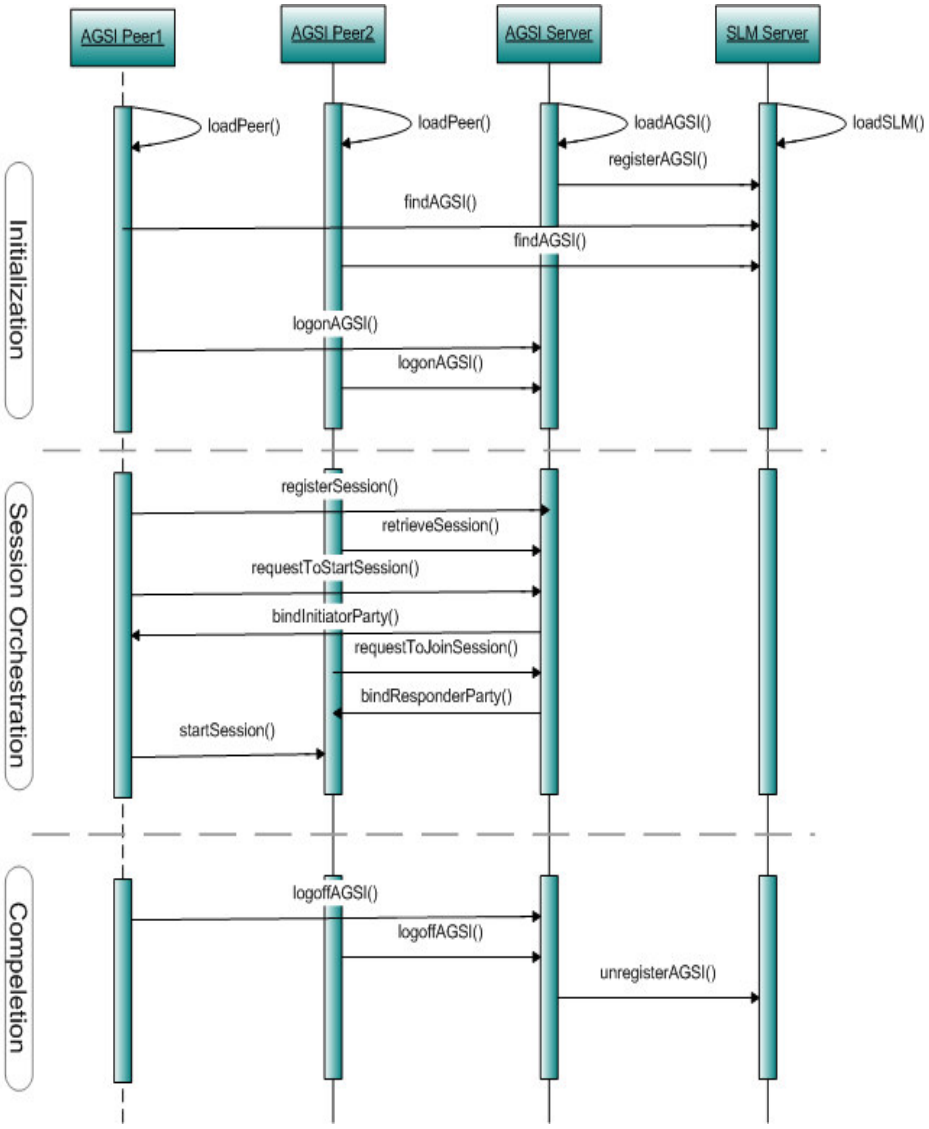

Figure 3.4-1: AGSI 3-phase Sequence Diagram

There are namely the initialization phase, the session orchestration phase and the completion phase. We summarize three-phase session operations in following Table 3-1.

Phase\Entity	AGSI Peer1 (Provider)	AGSI Peer2 (Consumer)	AGSI Server	SLM Server
Phase 1: *Initialization*	1. Load /Initialization 2. Look up SLM for AGSI groups 3. Log onto AGSI server that manages the AGSI groups	1. Load /Initialization 2. Look up SLM for AGSI groups 3. Log onto AGSI server that manages the AGSI groups	1. Load /Initialization 2. Advertise its offering services/applications onto SLM servers 3. Create member session entries	1. Load /Initialization 2. Register AGSI groups 3. Handling lookup requests from AGSI peers
Phase 2: *Session Orchestration*	1. Advertise own applications to AGSI server 2. Request to start one application session 3. Start providing session	1. Retrieve Session Directory from AGSI server 2. Request to join one application Session being authorized 3. Start consuming session	1. Add session information 2. Handle request session directory downloading for AGSI members 3. Create Session Controller (Stream Controller) from sessions 4. Bind session between Session Controller and Session Provider 5. Bind Session between Session Controller and Session Consumer	1. Handle requests from AGSI server for registering AGSI sessions
Phase 3: *Completion*	1. Logoff from AGSI server 2. Shutdown	1. Logoff from AGSI server 2. Shutdown	1. Clear member session entries in its session Pool 2. De-register self from SLM servers 3. Shutdown	1. Handle de-registration requests from AGSI servers. 2. Shutdown

Table 3-1: 3-phase AGSI Session Operations Description

CHAPTER 4
AGSI PROTOCOLS

In the previous chapter, we discuss the architecture of the AGSI framework. In this chapter, we give a detailed description of the protocols that make various parts of AGSI framework work properly.

4.1 AGSI MEMBERSHIP MANAGEMENT PROTOCOL

Under AGSI system, there are multiple kinds of entities like AGSI Group, AGSI sessions and AGSI peers. Each entity may have multiple instances and each of them shall be uniquely identified with an ID. It would be even more desirable that the ID used for one entity object can be self-descriptive or the ID itself can tell what type of entity associated with so that one can make out the type of the entity upon receiving the value of the ID.

In AGSI, we adopted using Globally Unique Identifier (GUID[9]) which is a 128-bit number that is generated based on network interface hardware address and a randomized number generated from the time the server was instantiated. There are five different kinds of entities existing in AGSI so far: AGSI Group, AGSI Server, AGSI Peer, AGSI Session and AGSI Role. As a naming convention, to generate a final ID, we add an entity type in front of the GUID as shown in

[9] GUID: it's a 16-byte field and originally coined by Microsoft.

Figure 4.1-1 such that it can be self-describing when the ID is passed from one point to another.

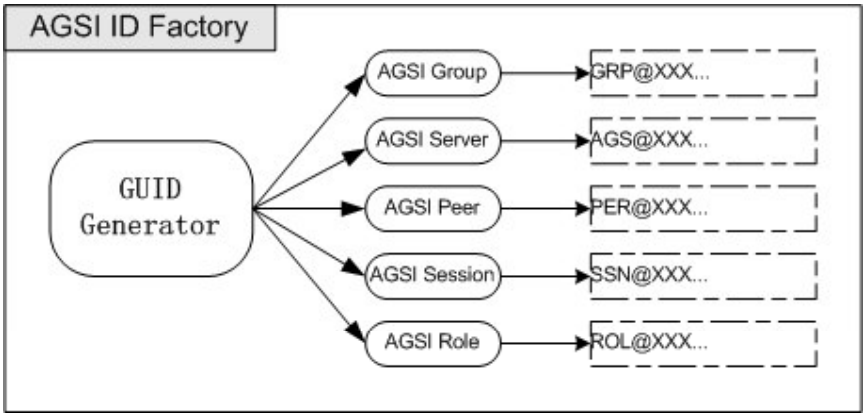

Figure 4.1-1: AGSI ID Factory and Its Naming Convention

To ensure security and integrity for a running application group system, the responsible AGSI server must invoke the AGSI ID factory and generate these self-describing identities for its managed entities. Individual party must get the identity from its associated AGSI server upon becoming a member of the group. Nevertheless, AGSI ID factory only generates and distributes these identities and does not store them at the ID factory.

Another possibility is to have a peer to generate an ID according to the same format as used in AGSI and claim that it has been distributed from the AGSI server. To avoid such a case and to offer a better security for the system, the identities generated from the AGSI ID factory shall be digitally signed with the private key of the associated AGSI ID factory. One member receiving that ID can render it for future use, which can eliminate the chance of forging a fake ID to cheat involved parties. The way of producing digital signature was discussed in section 3.3.2.

4.2 AGSI MEMBERSHIP ESTABLISHMENT PROTOCOL

AGSI group is about a set of users that belong to that group or are entitled to that group for the consumption or offering some applications within the group domain. AGSI membership establishment involves either of the following two cases:

- One is the membership establishment with some AGSI groups. As to which AGSI sub-groups, it is only a matter of further assignment based on the AGSI group membership. One user can establish her membership with more than one AGSI groups and one group member can belong to more than one sub-groups if necessary;

- The second is the membership establishment with AGSI sessions with the aim of providing access for each run-time application instance.

The difference between the two cases is the AGSI group membership is more static while the AGSI session membership is more dynamic. In the first case, AGSI group membership can be predetermined before initiation of any AGSI sessions. However, in the second case membership establishment is determined at a session lifetime and can be dynamically changed. It is important to have membership management at a session level since sessions also need protection from unauthorized access or consumption. The AGSI group membership is crucial for following reasons:

- Online users need to establish their membership with AGSI group servers and have their membership information kept there while they can roam around without losing their membership information and profile data that have been kept at their associated AGSI group servers.

- With AGSI group membership, it makes AGSI session membership establishment easier since AGSI group membership provides a means of scoping the users that have some common application interests.

- To manage and control AGSI group server, there need some security protection that shall be built upon the authentication and authorization of the AGSI group members.

To establish the membership for the above two cases, there should be some explicit processes, which are unavoidable. Below we discuss the two types of membership establishment for that can apply to the above two cases both.

- Approval-based membership establishment

- Open group membership establishment

4.2.1 Approval-based Membership Establishment

To establish the membership with one AGSI group or one AGSI session, one user must go through the approval process. In order not to restrict the forms of the internal mechanism for the approval process, we leave application authors to design and write an approval engine to determine whether or not to approve a user according to specific application requirements.

Nevertheless, we have already provided some basic approval methods in the framework like checking against a user's profile attributes to see if they match with the AGSI group or the session provider's requirements, meeting which the user can be allowed to create an entry in the system member entry director or session directory as shown in our AGSI architecture diagram.

Once a user is approved through the approval engine, the user will receive an establishment session token by which the user can further make calls with the AGSI servers for conducting session-related operations like publishing, initiating of a session and control of that session. Otherwise, if it is rejected, the user will not be able to establish the AGSI group membership nor the AGSI session access membership. The procedures in establishing with an AGSI group server and an AGSI session can be seen at Figure 4.2-1.

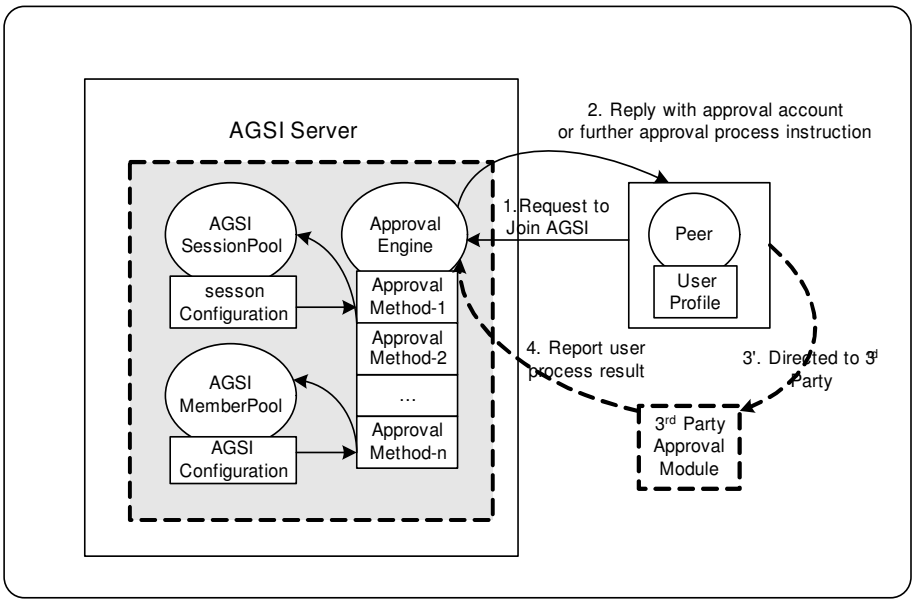

Figure 4.2-1: Membership Approval Process

From the above figure, we can see, the approval engine maintains a list of approval methods and the AGSI group server administrator can specify which method to be used to approve a membership establishing requests coming from the network. It can co-work with a third-party approval module for completing a membership establishment process. Once approved, a user will have an entry created in the AGSI member pool if It is for AGSI group membership creation, or a user will be able to access or create a session entry in the AGSI session pool so that she can further

conduct other session operations with a session token that has been granted after establishing the session membership.

4.2.2 Open Group Membership Establishment

It basically sets no restriction on recruiting new members. Almost anyone can become a member of the AGSI group except when the total number of recruited members of the AGSI group reaches its preset limit.

It is the extreme case and we normally do not encourage people to do so since it can bring out security issues like causing an overloading of anonymous member data and session data. Some malicious users could sabotage the normal session activities by accessing the session group freely while not behaving according to any rules.

However, the Open Group Membership Establishment is the simplest case, as it requires less infrastructure and deployment work and thus is more suitable for a temporary application group used for some small-scaled communities.

4.3 AGSI DISCOVERY PROTOCOL

Throughout AGSI, service discovery has been very important. Firstly AGSI peers need to discover their target AGSI groups according to their own requirements. Secondly they need to locate those responsible AGSI servers. Thirdly they need to locate the service/application providers. The overall process is depicted in following Figure 4.3-1.

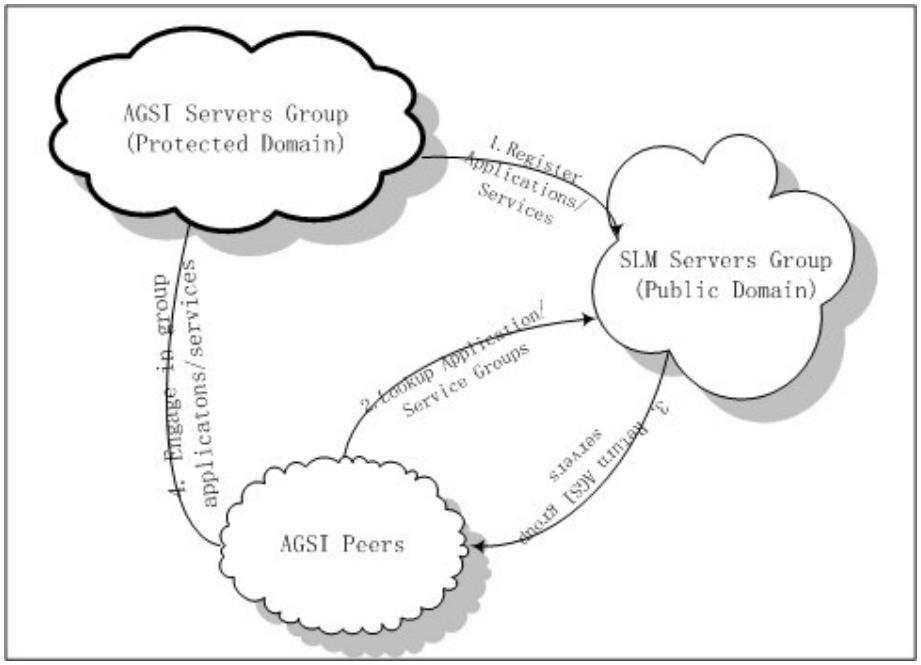

Figure 4.3-1: Service Discovery in AGSI

From the figure, we can see basically there are three types of entities in OCTOPUS application environment, namely SLM servers group, AGSI servers group and AGSI peers. Here we omit the CM (Connection Manager) part since it is transparent to this level as it is application-oriented. Following are the sequence of the activities incurred in AGSI applications/services discovery where the list below corresponds to the numbered circles shown at the above flow diagram:

1) Once an application group creator starts the AGSI server, she will register it with SLM server as a way of advertising the applications the group server provides. The registration is done through either multicast or unicast as the communication means provided by SLM system. One example of AGSI group advertisement can be found in Figure 4.3-2. Basically the group creator will advertise the common and public information about what an AGSI group will be doing. Of course, after publishing the group advertisement, the group administrator can still have chances to update the group advertisement for the specific AGSI group if it is desired. After the group recruits multiple members, some of them may be services or application providers or consumers and some may be both consumers and providers of different application sessions. Therefore the detailed internal services and applications will need to be further registered or published and looked up at the centralized AGSI server for efficiency.

```
<?xml version="1.0">
<agsi:group_ads>
    <group_guid>GRP@12345678....</group_guid>
    <group_name>E-learning Group</group_name>
    <application name="Java_Tutorial"  Desc="" />
    <application name="DBMS Tutorial" Desc="" />
    <application name="chat" desc="" />
    <service name="print" desc="" />
</agsi:group_ads>
```

Figure 4.3-2: Example of an AGSI Group Advertisement

2) AGSI peers connect to the network and send search request to the SLM system to get the remote reference of the responsible AGSI servers that host the wanted services or applications. After that, AGSI peers can proceed to contact these AGSI servers. The discovery is also done through multicast or unicast as the communication means provided by SLM system.

3) AGSI peers are directed to these AGSI servers and apply for the membership of the AGSI group as introduced in previous section. Once becoming a member of the AGSI group, it can access the protected domain that is governed by the AGSI server. But it does not mean the member would have full rights with that AGSI server and the member will by right obtain a common set of access rights specified by the AGSI group administrator. Furthermore, when other members of that group created some sessions, they would further specify the access rights for those specific sessions that may not be open to this member.
4) AGSI peers start conducting AGSI-group enabled activities like sharing some resources and applications or joining some applications provided by other AGSI group members. However those peers that don't belong to the AGSI group are prevented from accessing these resources and applications unless they are granted with access rights by the resources and applications owner or the administrator of this AGSI group. Within the AGSI group, peers can search for group members, sessions and other resources provided by individual members.

4.4 AGSI SEARCHING PROTOCOL

As we have discussed earlier, searching activities for resources or service/application sessions can happen at anytime among AGSI group members. To understand what are currently offered within the AGSI group, one peer may periodically search for the resources and sessions that are dynamically constructed or shared, on a polling basis. The reasons why we do not choose a push-based message delivery method are:

1) Members could log on and log off at an undetermined time and the AGSI group server can't hold responsibility for missing the shot of sending message to them.

2) It would be rather inefficient to have the AGSI server to send all of its updates to every member of the group since group members do not always want it and it is their desire to determine when to check for updates with the AGSI server.

Lastly, a peer may also want to search for the resources and sessions that are offered outside of the AGSI group or from the other AGSI groups that the peer has not yet established membership with.

To cater for the above needs required by a typical group system, we developed the following protocols to enable efficient search in the four different scenarios.

4.4.1 Search for Local Sessions

Sessions are run-time instances of applications or services. One session can involve multiple session providers and multiple session consumers. From the perspective of data communication, one session can be composed of data channels, audio channels and video channels. The data channels can be bi-directional and it can be a data stream

or strings of instant message. However, the audio channels and video channels can only be unidirectional due to the fact that in real practices one such a channel is associated with physical multimedia end-points that have one-way direction of data transmitting and are not likely to be changed once started.

Peers are attracted and recruited into the AGSI group for the purpose of consuming some sessions within the group. However to engage in one actual session, peers must get the knowledge of the online reference of the session providers and the session interfaces information for conducting consumption. Therefore, we design the protocol to allow peers to search for wanted session information as shown in Figure 4.4-1.

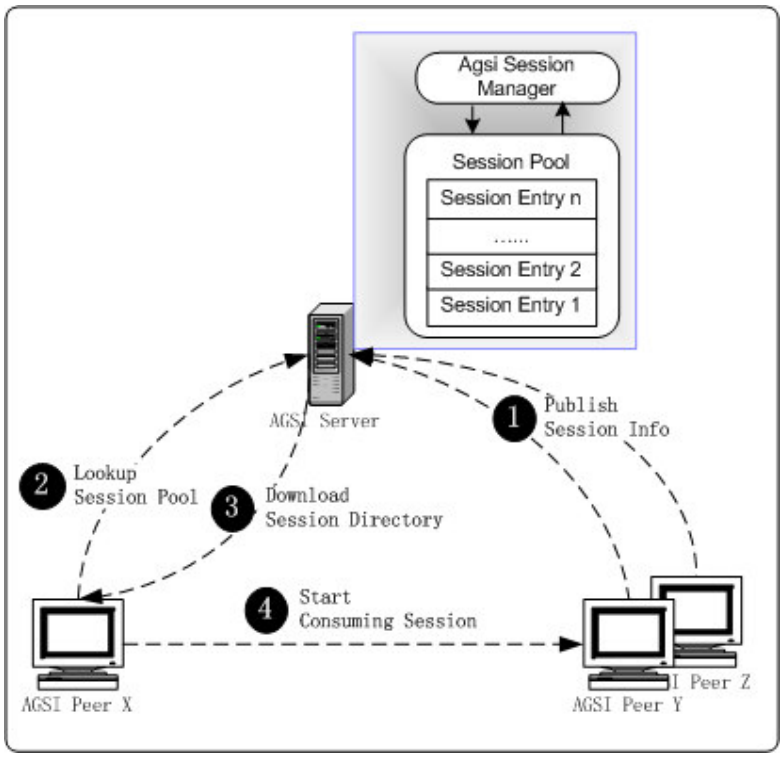

Figure 4.4-1: Search Local Sessions

The processes are listed as following:

1) AGSI Peer Y/Z publishes offering Session Info onto the session pool of AGSI server.

2) AGSI peer X connects to the AGSI group and looks up on the session pool at the AGSI server side.

3) AGSI peer X downloads the customized session directory that is generated by the session manager at the AGSI server.

4) Having the session information that includes the location of the session providers and the interfaces of the session, the AGSI peer X starts to connect with the session provider peers and consumes the session.

4.4.2 Search for Remote Sessions

Although AGSI creates the notion of virtual application group and restricts members to conduct activities within the group and prevents the outside members from accessing the resources, information and sessions of the protected domain, we do allow AGSI peers to discover the information of sessions that are provided and hosted in other AGSI groups. We treat the whole world as a single virtual world group and every AGSI group is a subgroup that belongs to that huge virtual world group.

Therefore, as long as the network infrastructure allows the transmission of the discovery messages across AGSI groups, we enable the discovery of the sessions that are conducted and managed within other AGSI groups.

After obtaining the search results, one AGSI peer can decide to join the other AGSI group and join the session or let the AGSI group join another AGSI group for that

session in a dynamic way. Session providers as well as the hosting AGSI server can decide to accept the joining request or not based on application policies.

Following are the list of activities that compose the flow to achieve remote searching for sessions throughout the AGSI communities.

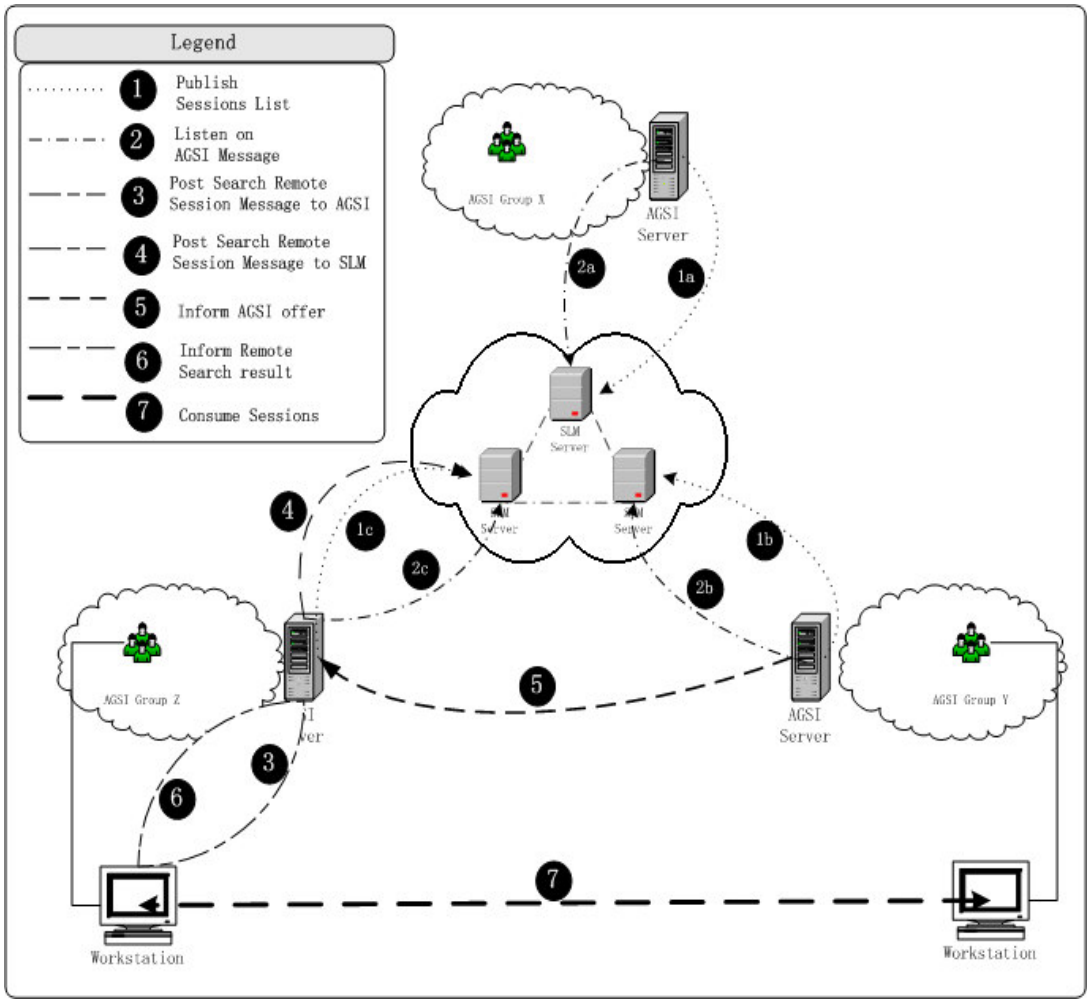

Figure 4.4-2: AGSI Search for Remote Sessions

1) AGSI servers publish their Group information and major session advertisement onto the SLM system but keep their detailed sessions information within their groups. (As noted as: 1a, 1b and 1c in the figure)

2) AGSI servers keep listening on the SLM in a periodic fashion and response to requests that are looking for some sessions that are hosted by the AGSI servers. (As noted as: 2a, 2b and 2c in the figure)

3) One AGSI peer tries to look up for some interesting sessions hosted in other AGSI groups by passing the requests to the directly associated AGSI server. The reasons why we do not encourage a peer to directly pass the requests to SLM system are: (a) it is better for a peer to engage in activities with one responsible AGSI group in one time unless the peer wishes to find or subscribe to other AGSI groups; (b) in order not to flood the open SLM system, all requests coming from one AGSI group shall be consolidated first at the responsible AGSI server which would handle them by forwarding only the new requests to the SLM system or response them immediately with the previous records that have been cached.

4) AGSI server consolidates the search messages and forwards only the new ones of them to SLM system to find other AGSI servers hosting the sessions as specified in the search message.

5) The AGSI server which hosts the sessions in demand keeps listening on the SLM system and would actively reply to the AGSI server that posts the messages for the sessions.

6) The AGSI server that received the answer would further notify the original requesting AGSI peer of the finding result.

7) After obtaining a necessary access right, the peer starts consuming the session with the session providers that are coming from other AGSI groups.

4.5 AGSI GROUP-TO-GROUP SESSION BRIDGING PROTOCOL

Within a single AGSI group, there can be multiple application sessions each of which may have different set of session members. Those different session-based member groups may interact with each other for sharing or exchanging some information and even a whole session. Furthermore, we treat the whole world as one single virtual group and under that world group there can be multiple AGSI groups. Between the AGSI groups, they may need to share or exchange some sessions as well.

To enable the interoperability among the various session-based groups regardless of what their belonging AGSI groups may be, we mandate the Stream Controller which resides in the AGSI servers to manage the interoperations including authentication, access control, bridging and dismantling of the session groups.

4.5.1 Introduction of OCTOPUS network-level group bridging

Group-to-group bridging is the most important and unique feature among all the group interoperations. It has something to do with the network-level multicast. Let us get some basic knowledge about network-level multicast before we introduce the group-bridging feature. To achieve network-level multicast for a single set of online users, we'll need to get each multicast user to subscribe to the network multicast group by sending out the requesting IGMP [22] message. When there are more than one such network multicast groups, let us say we have group A and group B, to allow members of group A to receive data that is from group B, we normally need to get each member of group A to further subscribe to group B by sending the requesting IGMP message. If group B wants to receive data from group A, we need to do the requesting job again reversely. If the number of members of group A or group B is huge, the job of merging the two groups will be rather laborious and could flood the network to some extent.

In OCTOPUS, one unique feature is proposed to help release the work of merging two network-level multicast groups. With OCTOPUS Connection Manager, merging two such groups needs only one time group-bridging call through modifying the internal functions of a network level router. Figure 4.5-1 shows us the process sequence in merging two session groups.

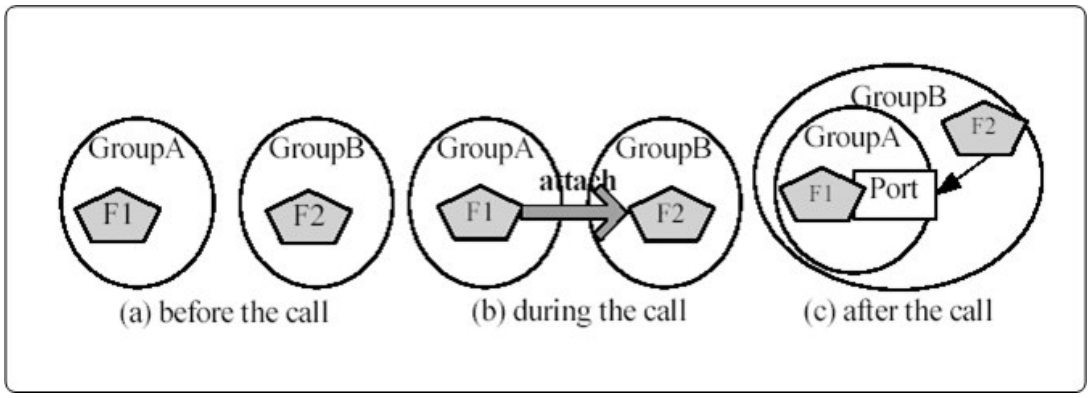

Figure 4.5-1: Session Groups Merging

Before bridging the two groups, each group contains one multicast flow endpoint. After sending out a group-bridging message to OCTOPUS Connection Manager, each data packet coming from group B will be forwarded to group A and group A will listen on the new port that has been assigned for receiving data flow coming from group B.

4.5.2 Bridging Two Session Groups at the Same AGSI Server

In many cases, two session groups that are created and managed by the same AGSI server will need such interoperation for allowing one group to access another. To achieve this, we continue to use the group-merging feature that is provided by OCTOPUS Connection Manager in AGSI framework as shown in Figure 4.5-2.

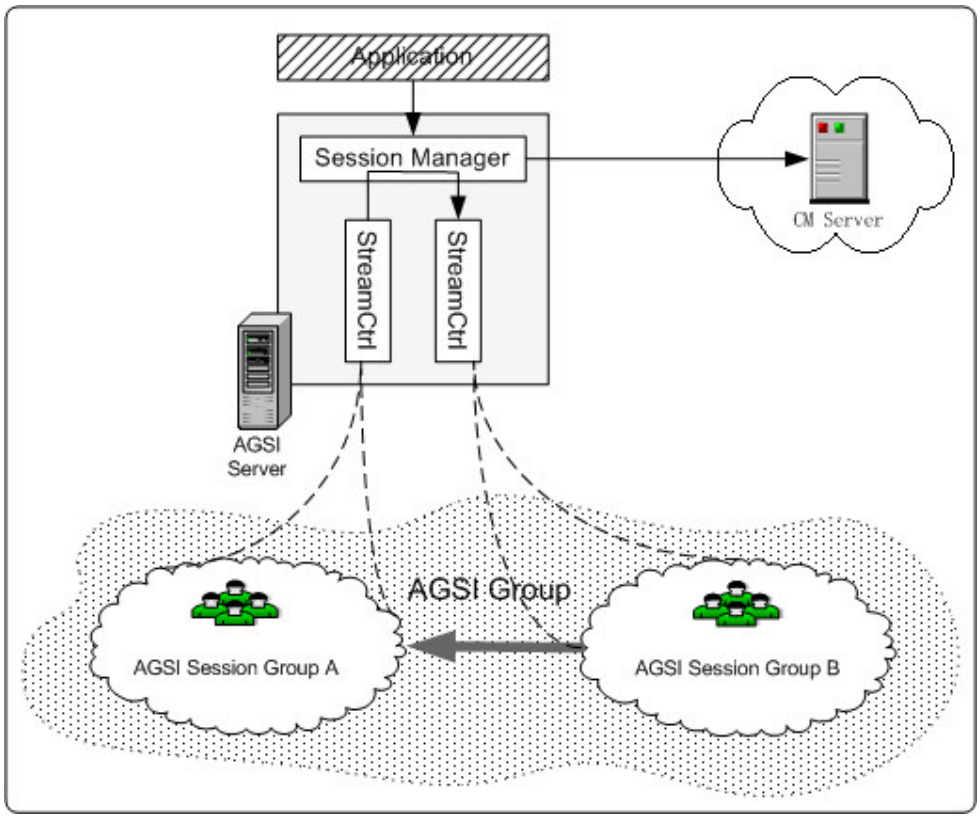

Figure 4.5-2: AGSI Session Groups bridging at the same AGSI server

From the above diagram, we can see there are two session groups managed by two Stream Controllers that reside in the same AGSI server and are managed by the Session Manager. To enable the access for the members of group B to group A, we need to invoke the group merging method at the CM servers by passing the multicast addresses of the two groups. Since the Session Manager can access the two Stream Controllers and retrieve the multicast addresses directly, it is very convenient for the session manager to invoke the group-merging call on the Stream Controller that stands for the initiating session group. Finally the group-merging call will get the Connection Manager to make the session group associated with another at the network level.

4.5.3 Bridging Two Session Groups at Different AGSI Servers

When the two session groups are coming from two different AGSI groups, to invoke the group-merging call, it requires a talk between the two responsible AGSI servers. The initiating party will firstly try to locate the AGSI server that is hosting the other session group to be bridged. Then it gets the multicast address of the session group from checking its Stream Controller. Finally it passes the pair of multicast addresses (IP address + Port) and talks to the CM server and asks for the bridging between the two addresses. Figure 4.5-3 shows the overall processes of group-to-group bridging.

Compared to the group-to-group bridging on the same AGSI server, here it requires the session manager in one AGSI to make remote call to another AGSI server that incurs the discovery of another AGSI server and authentication and authorization check when it goes through the remote procedure calls.

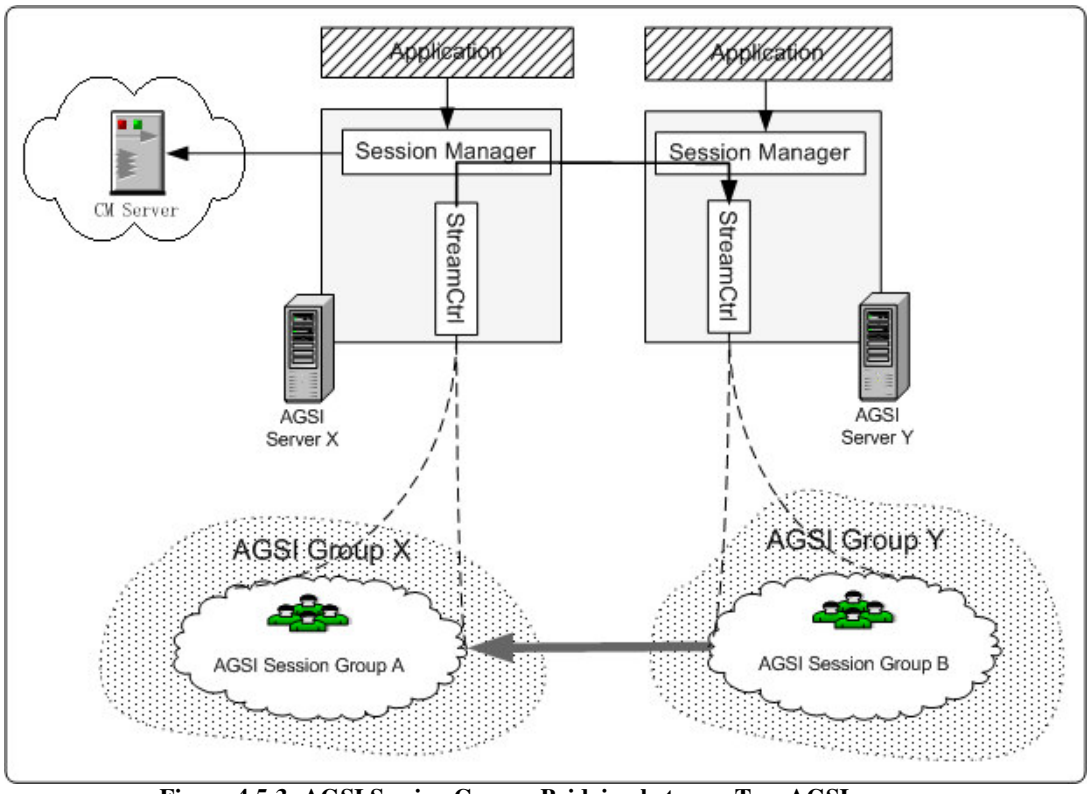

Figure 4.5-3: AGSI Session Groups Bridging between Two AGSI servers

4.5.4 Partial Member Joining in Group-to-Group Bridging

In a typical application scenario, as shown in Figure 4.5-4, the application programmer who runs and controls the application may want to limit the access of one session group from another session group. Also the session group manager may want to specify some specific a limited or partial set of members from another session group to access the session when the manager tries to make the aforementioned group-to-group bridging call.

So, how to enable the partial members access in the case of group-to-group bridging? This task would be impossible at the network level since an intentional user can always capture the data flow forwarded by the CM system. But we can however prevent the user from properly reading or understanding the content of the flow data by encrypting it without giving the decryption key. Furthermore, it is very likely that a normal application user will not be able to have the knowledge of what flows at which ports are flowing from other session groups without having a detailed group-to-group bridging flows spec. Therefore those flows being listed in the user's group-to-group bridging flows spec are visible flows. Others are invisible ones. However, an intentional user can find out the flow information by checking on the codec at the network transport level and report the results back to the application for a possible exploitation.

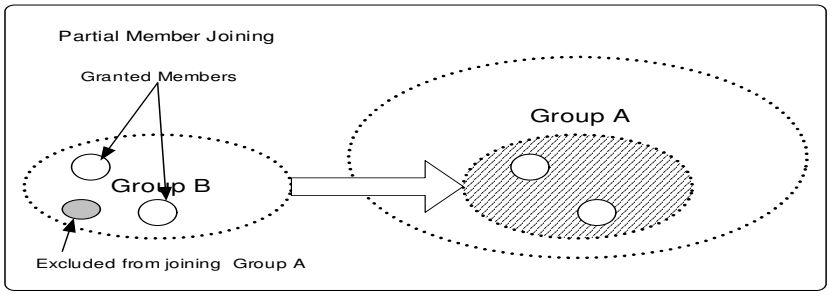

Figure 4.5-4: Partial Member Joining

Fortunately, the AGSI framework controls all of the activities of the Session Manager and application programmers will not be able to manipulate a Session Manager to find out all those visible and invisible session flows. But to allow a partial set of group members to be able to consume the session flows, these members must be notified about the flows information including the flow port numbers and the flow session identities and even the data decryption key if there is any. Having the information, i.e. the flows spec, one member would be able to read the flows data and join the sessions properly. In contrary, those who do not have the right information will not be able to join the sessions even though they may be still unwittingly receiving the flows that are invisible to them.

4.5.5 Cascading Group-to-Group Bridging

It is not atypical to encounter such a scenario where cascading session groups get bridged in a sequential fashion. Looking at the network transport level, we will find out the data flows go in the way as shown in Figure 4.5-5.

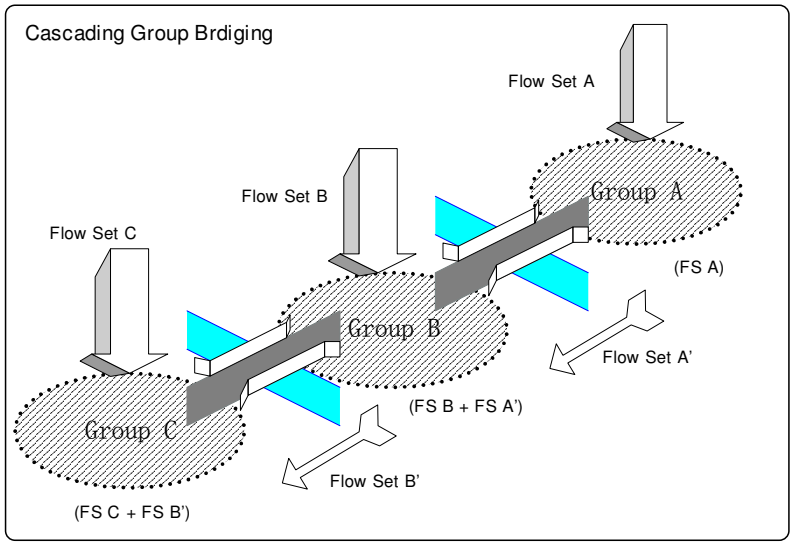

Figure 4.5-5: Cascading Group Binding

In multimedia communication, we treat a single communication channel (represented by a multicast IP address and a port) as a flow and multiple such flows compose a

stream. A session may be further composed of multiple streams. In other words, a session group can have multiple multicast addresses and ports. However, as an engineering practice, we choose a single multicast address for all of the flows within a session group but use several different ports for different flows. In some cases, we could use the same multicast address and port for more than one flow but with different flow session identity for each flow.

As we can see from the above figure, there are three such session groups. The session group A has a flow set A that is an aggregated set of flows denoted by (FS A) that is a set of elements each having the data format: (Multicast Address, Port Number, Flow Session ID), so is it true for group B (FS B) and group C (FS C). Group A is bridged with group B and a smaller set of (FS A), denoted as FS A', flows into group B. Then the group B will have a new set which is a combination of the two sets as denoted as (FS B + FS A'). Furthermore, group B and group C are as well bridged and group C will have a new set denoted as (FS C + FS B'). Let us take a note here: (FS B') is a smaller set of (FSB + FS A') thus it would contain some flows from (FS A') and (FS A). That is, group C can contain flows coming from both group B and group A.

Will it cause unnecessary problem like one peer would sneak into the flows even without permission? Or will it cause one peer to be flooded by the excessive incoming packet flows?

Thanks to the introduction of Session Orchestration Service that is a part of AGSI framework, these phenomena would not occur. Session Manager represents the Session Orchestration Service. It actively intervenes the flow-level sessions consumption by checking the flow session identities to see if they are in the permit list, or the flow session access control list. If they are permitted the application tier can

receive flow sessions otherwise the flow sessions would be immediately discarded or ignored. This kind of checking is applicable to every session group.

Finally let us look at the Figure 4.5-6. It shows how the Session Manager handles jobs like group-to-group bridging and partial group members joining another session group. The key part is to maintain a correct session access control list for the stream controller so that it can notify the right members about the sessions they are allowed to access. The session access control list contains the IP address, port number and session ID and the decryption key if the flow data has been encrypted.

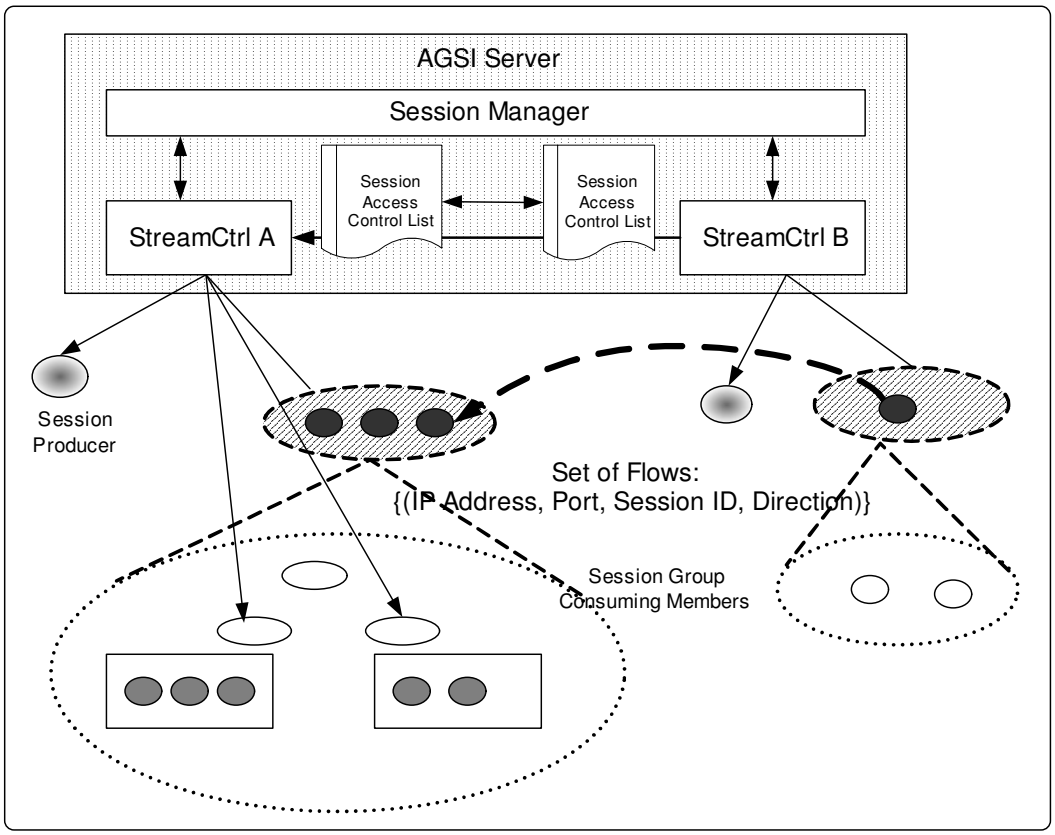

Figure 4.5-6: Group Bridging Access Control

CHAPTER 5
AGSI IMPLEMENTATION AND EVALUATION

This chapter presents the implementation details of AGSI core components and services. The implementation language adopted in both AGSI and OCTOPUS is Java due to its platform independence as we are dealing with heterogeneous environment. The key Java technologies used in AGSI and OCTOPUS are J2SDK 1.4, JMF, Java RMI and Java JINI technology.

The overall AGSI implementation can be divided into two parts: one is AGSI Server part and the other is AGSI peer part. AGSI server is an independent server application that runs to serve requests from other peers like AGSI peers or even other AGSI servers. AGSI peer wraps OCTOPUS multimedia and networking capabilities inside and add a set of CSCW features that enables it to talk to AGSI server and other AGSI peers. Every AGSI peer has a heartbeat function within that keeps checking on the network environment and maintains the connectivity with associated AGSI servers.

5.1 AGSI SERVER DESIGN AND IMPLEMENTATION

AGSI server plays a key role in AGSI paradigm. From Figure 5.1-1, we can see all the components that compose the AGSI server. One AGSI server connects a database server for storing basic information like member profile data and group details and group structure data. AGSI server also has its own profile that is kept in a XML file.

One can modify the XML file and re-run AGSI server with different system settings. All the details about specific components will be introduced in following sections.

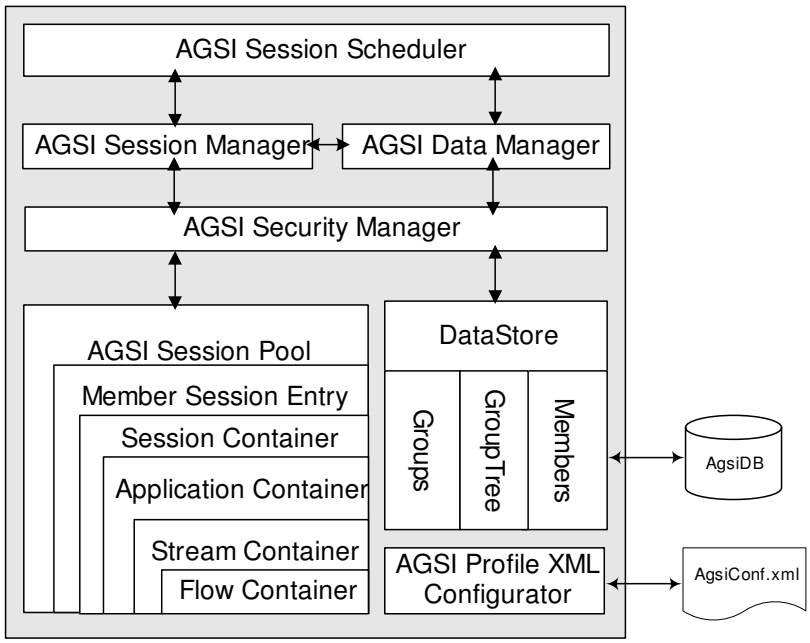

Figure 5.1-1: AGSI Server Diagram

5.1.1 AGSI Session Scheduler

It is an independent thread running with AGSI main thread. It basically works on schedules sessions like starting, stopping and resuming a session as scheduled. It also does the house-cleaning job for AGSI session pool. If a member stops its session on one AGSI server without notifying it, the AGSI server will still keep a member session entry for the member for a while. The AGSI session scheduler will check if the member with that entry is alive and valid. If not, it will then remove the session entry thus release the memory from storing useless data.

From the above diagram, we can see AGSI session scheduler sits on top of AGSI session manager and AGSI data manager. Therefore, all the calls from the scheduler will have to be passed to one or both of the two managers to complete its tasks. By doing so, it makes the whole system modularized and flexible. It also leaves space for

the session scheduler to have new features but based on the exposed functionalities of the two managers.

5.1.2 AGSI Session Manager

It handles everything on sessions that are runtime instances of AGSI applications. It works together with AGSI data manager to accomplish the majority of internal and external jobs for AGSI server.

AGSI session manager handles requests from AGSI peers like creation of member session entries and adding member application session advertisement information. It works on directly with AGSI session pool that is a common repository for AGSI group. All the session information for AGSI group will be managed by AGSI session manager and kept in AGSI session Pool. AGSI session manager is also responsible for creating the session controller that is embodied in OCTOPUS Stream Controller which helps initiate various hosted sessions. The session instigator or session administrator can operate on the session controller to do some session-related operations like binding of both session parties and start/stop of the session and bridging between two session controllers. To balance the workload incurred by the session control operations invoked from a great number of session members, we allow creating the session controller at a third-party host machine, for example, the session provider that initiates the session. AGSI session manager will record the remote reference of that session controller and pass the information to the remote session controller for conduction of the session.

Another important job AGSI session manager does is access checking. When a member comes to look up some sessions she is allowed to access. AGSI session manager will base on the session access control list to filter out all the unauthorized

sessions from the member and return all the sessions that have been authorized to the member.

5.1.3 AGSI Data Manager

It does basic data processing like adding, updating and removing a group or a member record. It talks to some database that in the case of our implementation is MS SQL Server 2000 database. But it can talk to other kind of database as long as it provides the common database functionalities. Through the connection part of AGSI Server XML profile, we can easily change the connection to other database without changing much code at the AGSI session data manager.

The reasons why we still use a physical data store to contain some information rather than to put them all in AGSI server's memory are that: 1) We need a consistent place to store those static data like member profile data, group data and group structure data, while AGSI server memory can only store volatile data like runtime session information 2) Implementation of AGSI group can be further separated from the data storage, which means multiple AGSI groups can host their data on one single database.

5.1.4 AGSI Security Manager

It provides security support from the session level. Every remote call from other party to its AGSI session manager or AGSI data manager will be subject to security check that is done by AGSI security manager.

AGSI security manager checks the given signature in every remote functional call from one member with its public key that has been stored in the member's session entry. If AGSI security manager find the signature is valid and represent the member, it will then allow the call proceeds otherwise it stops the remote call immediately.

AGSI security manager also provides some helper functions like generating of public and private key-pair, generating of signature for individual members. A member can obtain the key-pair when it gets subscribed to the AGSI server and can keep a copy on its local public and private key files.

5.1.5 AGSI Members Pool

It is a data structure purely to hold all the member session entries at AGSI server when members newly logs into the AGSI server. It keeps the very basic information about a member like the online status. With that, one can easily become aware of his or her other group members.

Before one member logs off from AGSI, the member will notify the AGSI session manager and have it to empty the corresponding member session entry in the member pool. If the member does not do notify the server, the session entry will also be automatically removed by AGSI session scheduler after no reaction from the member's side.

5.1.6 AGSI Sessions Pool

It is a data structure that is used to contain all the session information for one AGSI group. The AGSI session pool is physically made of an AGSI session container that is hash table. It further maintains a list of AGSI application specification that records the information about one specific session. The whole data structure can be seen at following Figure 5.1-2.

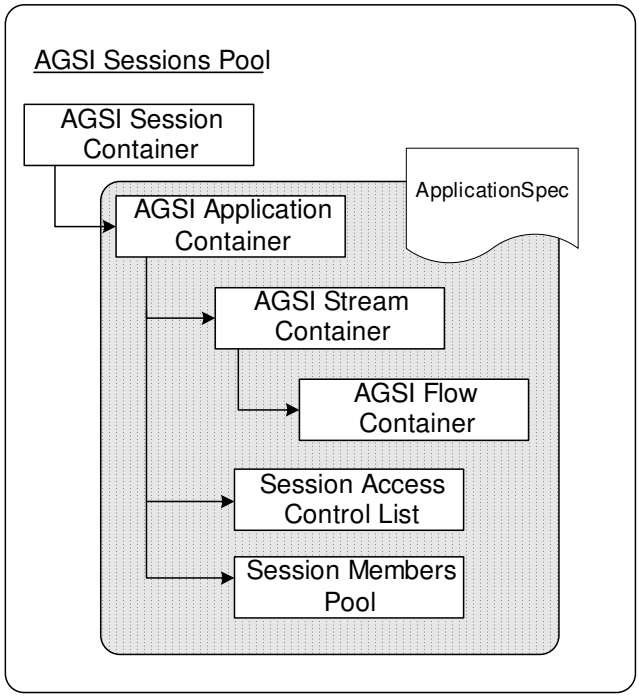

Figure 5.1-2: AGSI Session Container Data Structure

From the diagram, we can see that different containers are structured in a cascading manner but in different levels. Firstly the session container maintains a list of application containers that keep all the information for various applications, i.e. the application spec within one AGSI group. Then the application container maintains a list of stream containers that keep stream information of one application. At the bottom level or under the stream container is the flow container that keeps the information about a flow, be it one audio, video or data flow.

In a nutshell, application spec for each session is the key data structure that is indispensable for running and managing a session. The original application spec may come from a session provider or a session controller. One application can hold multiple application components and many data or streaming flows in its lowest level. For example, we can put a chat component and audio/video conferencing component together within an application spec and have our AGSI to initiate and manage the

whole session starting from its inception to its very end. The application spec can be captured in a XML file and transferred in XML or data object format.

5.1.7 AGSI Data Store

It is basically a physical database where keeps the information about member profiles and group profiles and group structure.

5.1.8 AGSI Profile XML Configuration

It loads AGSI server configuration data from the AGSI Server Configuration XML file. It can be further used by other parts of AGSI server. One can modify the XML file and re-launch the AGSI server to have different kind of environment settings.

5.1.9 AGSI Server classes diagram

We list the major components' classes diagram in Figure 5.1-3.

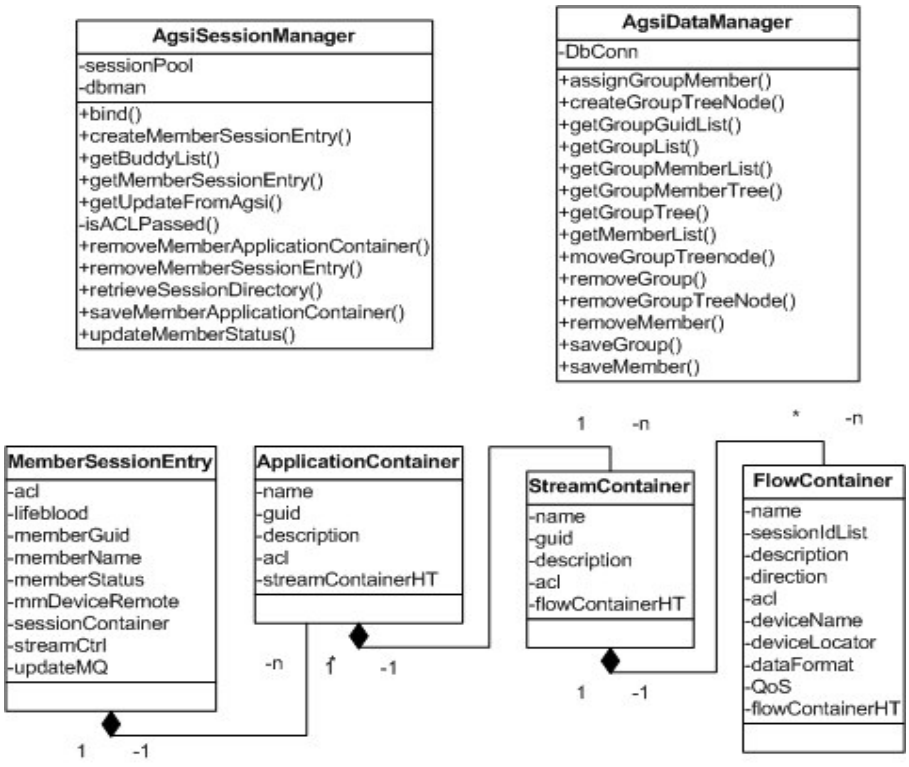

Figure 5.1-3: Classes Diagram for AGSI Server Major Components

5.1.10 AGSI Server Main Flow Diagram

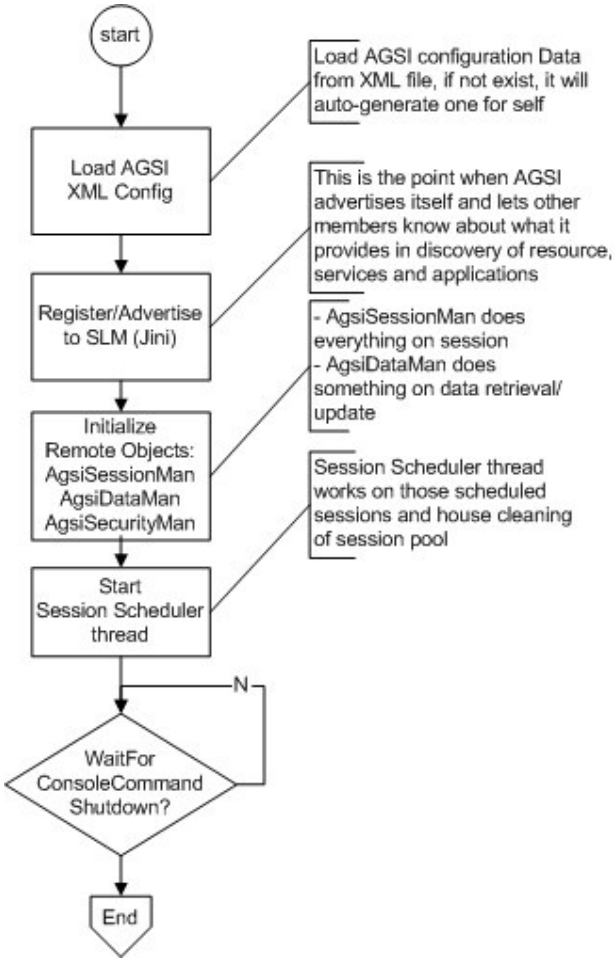

Figure 5.1-4: AGSI Server Bootstrapping Flow Diagram

To launch an AGSI server (Figure 5.1-4), the following tasks have to be completed:

1) Load configuration xml data;
2) Register self with SLM server;
3) Create three manager components: session manager, data manager and security manager;
4) Start session scheduler thread;
5) Run a loop waiting for console command.

5.2 AGSI PEER DESIGN AND IMPLEMENTATION

As one integral part of AGSI system, AGSI peer as shown in Figure 5.2-1 provides middleware support for various applications. Application programmers can just include AGSI peer to write distributed peer entities. All those features like interacting with remote AGSI servers and establishing sessions have been enabled within AGSI peer's middleware part.

Like AGSI servers, AGSI peers are also identified by a GUID but with a different prefix tag: "PER". AGSI peer middleware contains multiple components within but only two key components are exposed to the application layer. One is a single AGSI proxy that manages internally multiple AGSI agents. Another is AGSI peer heartbeat that is an autonomous thread running along with a peer's main thread. It will periodically check the connectivity of remote AGSI servers and get updates from them. AGSI peer has included previous OCTOPUS major component Multimedia Device and thus has the capability of conducting multimedia sessions.

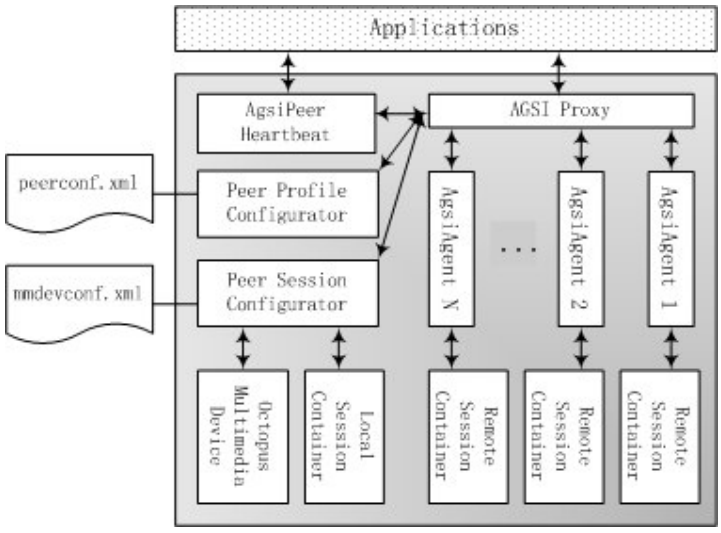

Figure 5.2-1: AGSI Peer Diagram

5.2.1 AGSI Proxy

It is a wrapper class that contains OCTOPUS multimedia device, AGSI heartbeat and AGSI agents. It is responsible for spawning AGSI agents according to the peer's XML profile configuration. It does everything that is common to all AGSI agents. Applications will only need to talk to AGSI proxy to access AGSI agents or the heartbeat thread. In another word, to have AGSI peer's functionalities, one peer application just needs to include AGSI proxy and it will suffice to do all the jobs that are required for interfacing with AGSI.

5.2.2 AGSI Agent

It talks to one AGSI server that is associated with. To engage in one AGSI group, a peer needs to get subscribed with one AGSI server and one AGSI agent will be needed to operate with the server.

5.2.3 AGSI Session Configuration

It is a useful tool that it helps construct the content for a multimedia device. It also helps construct a session container or an application container and even the mirror set of an application container that contains the stream and flow settings for the other party. One can create a XML file and use it to let session configuration construct the session container.

5.2.4 AGSI Profile XML Configuration

It helps load a peer's profile data from one XML file. It also helps construct its own public key and private key from local public key file and private key file.

5.2.5 AGSI Session Container

Its data structure is the same as that of AGSI Session Container in the Sessions Pool as shown in Figure 5.1-2, which is also to maintain a list of Application Spec. But in a peer's side, there are two such session containers. One is created for storing application spec of local sessions, i.e. all the sessions that are initiated by peer itself. Another is created for holding application spec of sessions that are initiated by other parties from remote. Each AGSI Agent maintains the list of remote application spec that is updated with the session directory downloaded from its corresponding AGSI server.

5.2.6 AGSI Peer Heartbeat

Every peer has a heartbeat thread that does important jobs. As shown in Figure 5.2-2, the heartbeat function will repeatedly check the connectivity of associated AGSI servers to see if there's any change. The changes can be of many types and are extensible for further development. The current implementation checks on the group member status and session directory.

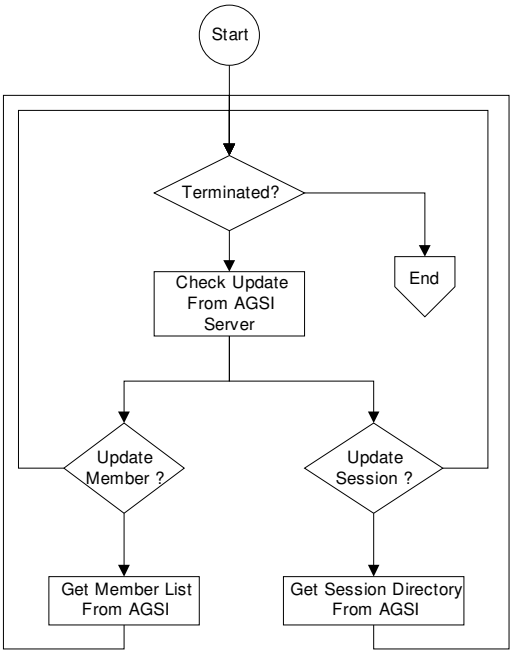

Figure 5.2-2: AGSI Peer heartbeat thread Flow Diagram

Once the heartbeat function retrieves some updates, it will talk to application layer to update the UI if necessary. On the other hand, AGSI server will be able to know the peer is alive and functioning properly and thus would not clear the peer's member session entry from the session pool. Otherwise, once the AGSI server lost contact with a peer, i.e. not receiving any message from the peer for a while, the server will clear the entry within its session pool.

The reason why we let AGSI Peers periodically poll the corresponding AGSI servers for the checking of any new or updated sessions is that, AGSI servers and the session providers can't know beforehand if the wanted session members are online or not and in some cases It is impossible to predetermine the session access members list. Therefore it is quite convenient for individual peers to check with the AGSI servers in an active manner. Since not all the allowed members are always online in the same time, checking from peers can therefore save the network traffic compared to that of broadcasting session information to all of the session members.

5.3 AGSI GROUP-2-GROUP BRIDGING/DISBANDING

The group-2-group bridging/disbanding means an application group A (hosted in AGSI group server) can share its applications to another group B (hosted in another AGSI group server) without having the members of the group B to be enlisted in group A. The original design of AGSI relies on CMS (Connection Management Service) to provide the group-to-group merging/disbanding services. In the case of group-to-group merging/disbanding, users can issue commands to CMS system and realize the group-to-group merging/disbanding. The implementation in AGSI part is to provide a service that can handle requests like merging and disbanding of two multicast groups. The service can invoke the CMS service subsequently. Because CMS also leverages an overlay networks and can cater for interconnected multicast islands, AGSI can rely on that feature and send commands to CMS without caring about the underlying network connectivity as well as multicast support.

The actual implementation of AGSI did not leverage on CMS due to some technical difficulties in integrating AGSI and CMS that would require substantial re-coding effort to overcome. Instead, an emulation of CMS was put in place. In the emulation, a centralized registry server was created to provide information of group multicast memberships and information of relay agents. A group merging operation (similarly for group disbanding operation) invoked by a group's representative to his AGSI server will result in the latter to consult the registry through the local relay agent (which can be run within the same machine of the AGSI) for information of the corresponding group addresses and the other relay agents. Then the local relay agent makes the necessary unicast connections to these agents (please refer to next paragraph for a detailed description of this operation) for relaying of data.

Comparatively, this emulation is a centralized approach (the registry) for any AGSI server to locate other peer AGSI servers before any data transfer can take place. On the contrary, the CMS is a distributed approach. Requests of group-merging/disbanding sent to CMS do not rely on any individual server. The whole virtual layer can respond to the requests and act accordingly.

The implementation of this new approach is use Java RMI service and JMF media capability to do the remote invocation and streaming forwarding. This approach does not comprise the effectiveness of the whole AGSI system in following reasons:

1. This approach applies the same idea that is used in CMS for enabling group-to-group merging/disbanding.

2. AGSI is a system to provide group and session management, which aims at the session and application level. On the contrary, Group-to-group merging/disbanding is working at network and transport layer, which is supposed to be the work of CMS. Changing the implementation in the lower layer should not affect the higher layers as long as the interfaces are fixed.

In order to demonstrate the group-2-group bridging/disbanding feature in AGSI application, a special service has been developed to achieve the group-2-group bridging/disbanding. It is called session reflector in AGSI system that can reflect one group session onto another group. It is implemented almost in the same fashion practiced in CMS implementation except that AGSI reflector service does not rely on any overlay network service. AGSI reflector service talks directly to network layer by linking two multicast-addressed subnets together.

When a streaming application sends streams to one multicast address, the data will be forwarded to another multicast address. However, it is not possible to directly multicast streaming data from one network to another when multicast routing is not available between the two networks. This has been well addressed in AGSI reflector service by setting up two agents in two different networks. Both agents and their group information are registered in a common registry. When the group bridging command is issued from one subnet to another, the agent from the first subnet consults the common registry and get the agent unicast address and associated its group multicast address from the second subnet. The first agent also listens on the local multicast address and notifies the second agent and forwards the data to the other subnet via a unicast channel. Meanwhile, the agent from the second subnet receives the incoming unicast streaming notification. It starts listening on the unicast channel and forwarding the incoming streaming data immediately to the local multicast address. At this moment, the source session has been bridged to the destination session. In order to disband the bridging links, just disband each link during the path in a reverse sequence. Below diagram Figure 5.3-1 shows the activities flow to complete group-to-group bridging.

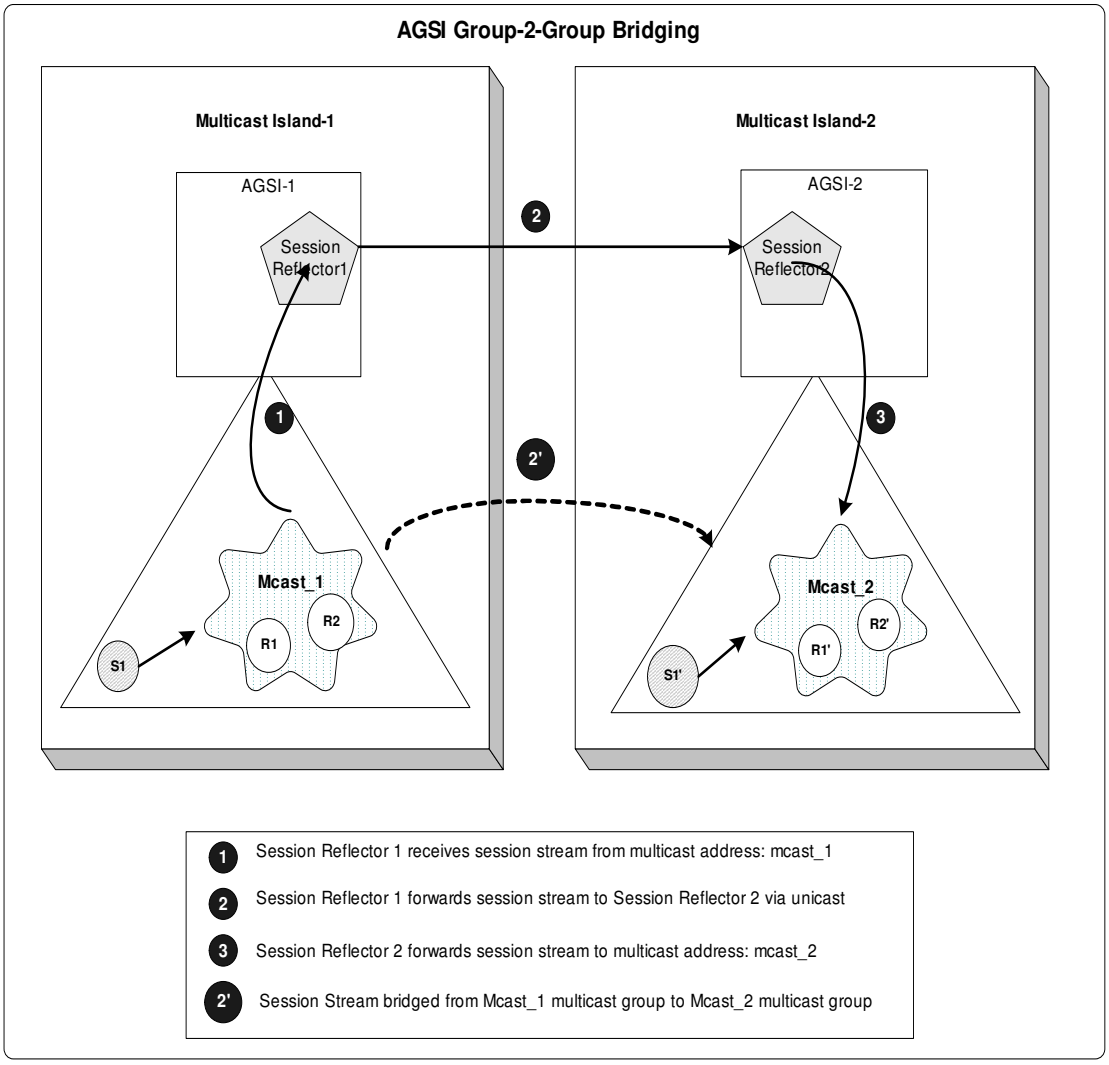

Figure 5.3-1: AGSI Group-2-Group Bridging Diagram

5.4 AGSI TEST-BED

The technologies used in AGSI system design come from the fields of distributed system, database and XML. The primary AGSI performance metric used here is the latency in performing one or more operations. The justification for using latency as a performance metric rather than throughput is that collaboration is interactive and requires fast response time. Low latency implies fast response time whereas high throughput does not necessarily mean fast response time. The experimentations conducted on the test-bed can be divided into two groups: one is general session operation and another is AGSI group-2-group operation.

5.4.1 Test-bed Configuration for general session operation

Below Figure 5.4-1 shows the test-bed that has been used for general session operations and Table 5.1 shows the details of the hosts' and the network's specifications. From the test-bed diagram, we can see there are total three sub-networks, two of which are wired LANs and another is wireless LAN. The wired LANs have 100 Mbps connection and the wireless LAN has 11 Mbps connection rate. There are AGSI servers, AGSI peers and SLM (Service Locating Manager) Server that exist in the whole network. Our objectives of performance testing here are to test the performance of AGSI servers and the interaction performance among various peers and servers.

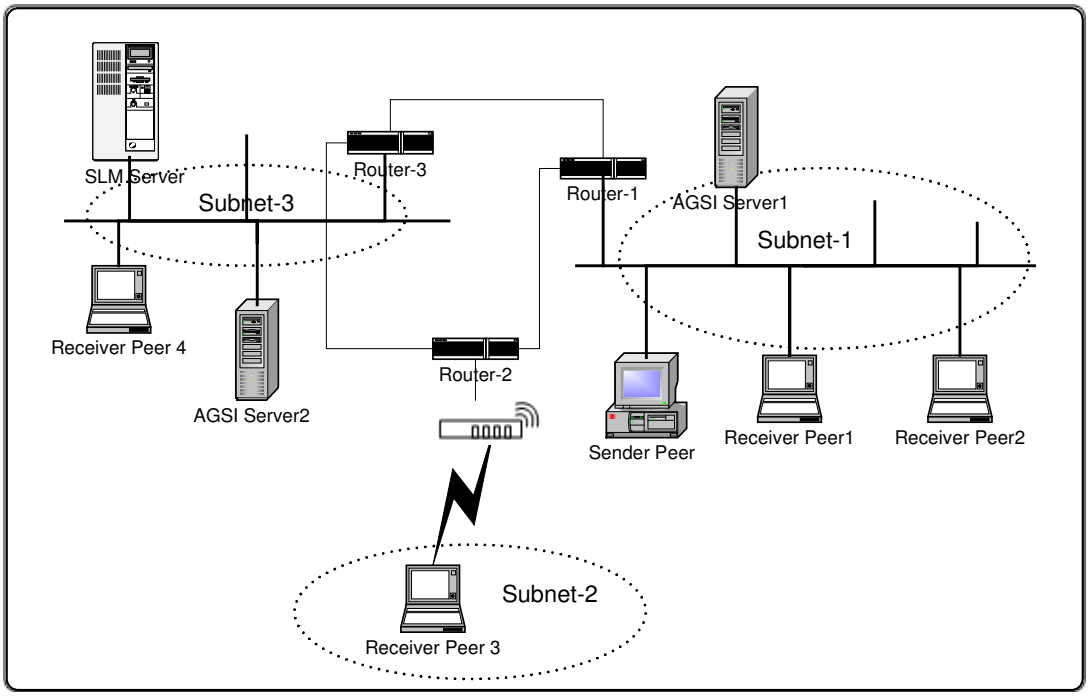

Figure 5.4-1: General Test-bed diagram

5.4.2 Test-bed configuration for AGSI Group-2-Group operation

In order to test the functionality and performance of the group-2-group merging/disbanding service, experimentations have been conducted in the same OCTOPUS test-bed as shown in Figure 5.4-1. The experimentations were conducted in two network scenarios: (a) Networks composed with m-routers (multicast routers). This scenario is to study the processing overhead of the AGSI server and the individual component, and (b) Networks composed with emulated CMS that provides multicast group-2-group bridging/disbanding functionality. The details of the scenarios are explained below:

- AGSI group demonstration with native IP multicast-enabled routing service

Figure 5.4-2 shows an enlarged portion of the whole network. It is composed by subnet-1 and subnet-3. The detailed IP configuration information is shown in this diagram. With m-router service running in every involved Linux router, the same

multicast stream can be transferred through each router and thus able to reach whichever clients attached to those routers forming individual networks.

The experimentation has been conducted as following: in the router-1 attached network (net-1), an AGSI group server and a AGSI peer were invoked to run in the attached net_1 network; in the router-3 attached network (net-3) another AGSI group server and one AGSI peer were also running in the attached net_3 network. Both peers logged into their respective AGSI groups in different networks (e.g. group 1 in net-1 and group 2 in net-3) and started sharing and consuming multimedia applications. It was tested with success in transmitting multicast streams from one peer to another via routers that are installed with m-router service and are thus able to pass through multicasting data. In this setup we assumed all members of an AGSI group are located within the same LAN (e.g. net_1 and net_2). This is not realistic but is sufficient for the measurement of the processing overhead of the AGSI server.

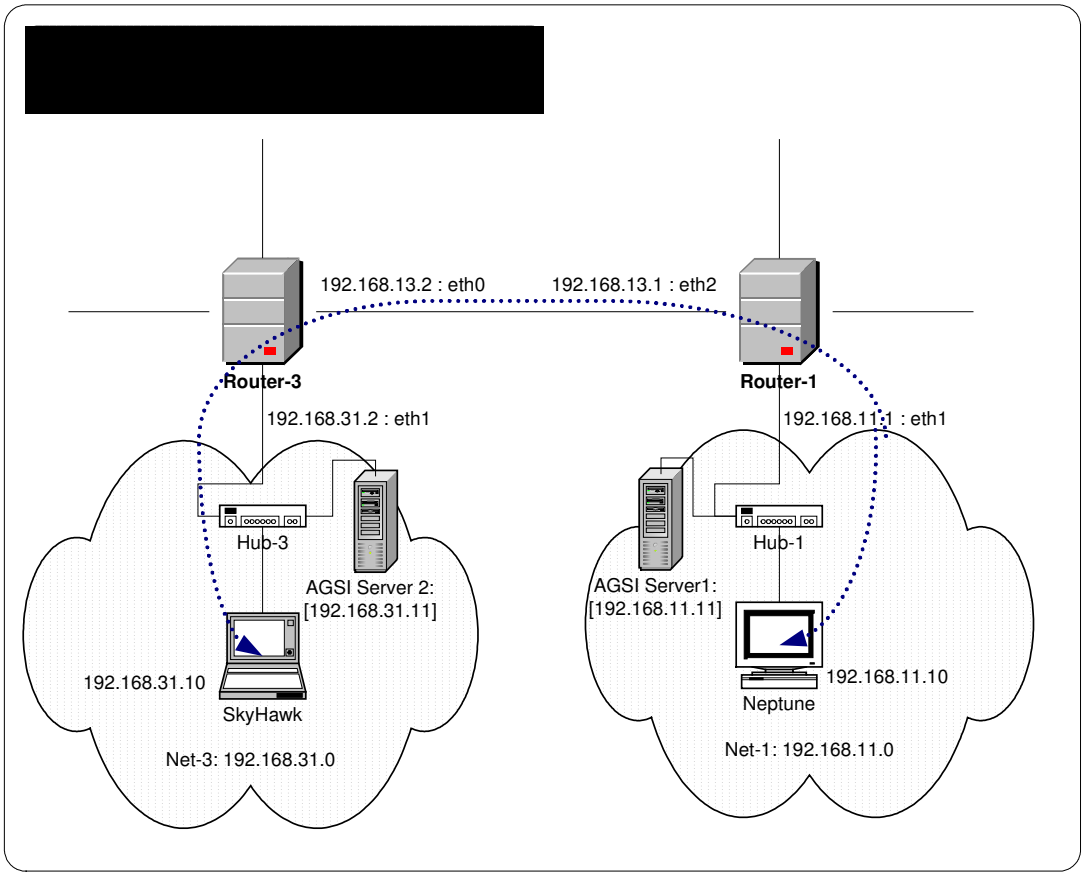

Figure 5.4-2: AGSI Group-2-Group Test-bed Diagram

- AGSI groups demonstration with emulated CMS

There are numerous existing Internet routers that could not support multicast network service (most of which are having their multicast capability disabled for security reasons). To overcome this restriction, a new idea was proposed and implemented in OCTOPUS that extends CMS [6] to multicast islands (the unit of intranet that supports IP multicast; a LAN segment is an example of such island can exists at the lowest network hierarchy). The extended CMS makes use of a 3rd-party overlay enabling software to create the multicast overlay network and implement the application group-2-group merging/disbanding functionality. In this demo, a simplified emulation has been developed to enable the testing and evaluation of AGSI services. The architecture

for this emulation known as the AGSI session reflector has been discussed in the Section 5.3. The operational flow for this experimentation is shown in Figure 5.4-3. In this diagram, AGSI common registry was used for registering network references for AGSI servers and multicast group names. Thus, the common registry can be used for looking up the AGSI servers that manage certain multicast groups.

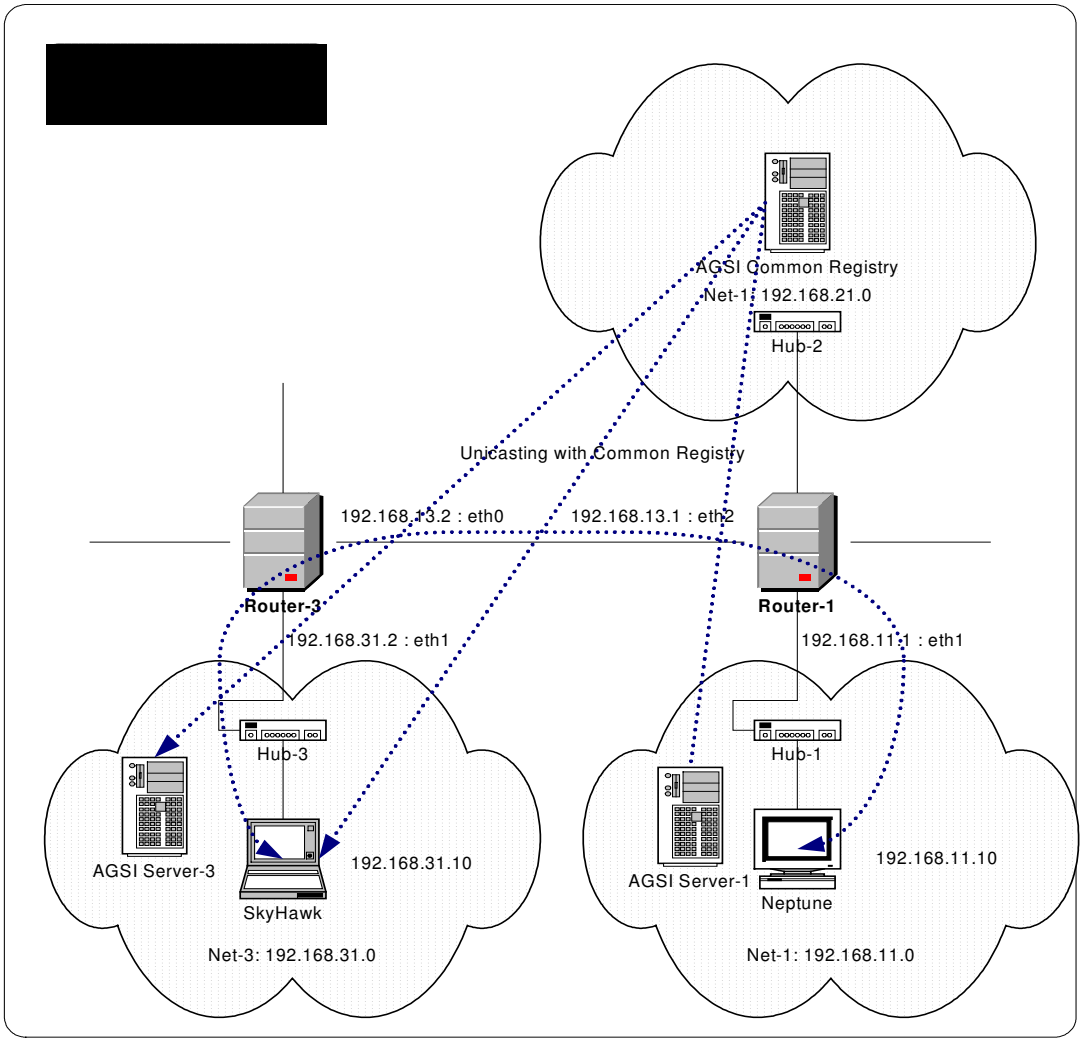

Figure 5.4-3: AGSI Group-2-Group bridging Using Emulated CMS

5.5 EVALUATIONS

This section evaluates the experimental results of various testing scenarios introduced in Section.5.4.

We used latency as the metric to conducting our testing experimentation. The common method used for evaluating the latency is shown in Figure 5.5-1. We take the time cost as the major performance evaluation metric to evaluate the efficiency and latency of the AGSI implementation in its various session activities. As AGSI is a system whereby all of the distributed AGSI peers communicate with each other through the support of the AGSI group servers. Given a set of AGSI peers and one AGSI group, there can be various combinations of communication activities. However, all of the communication activities are composed of point-to-point communications as shown in following diagram.

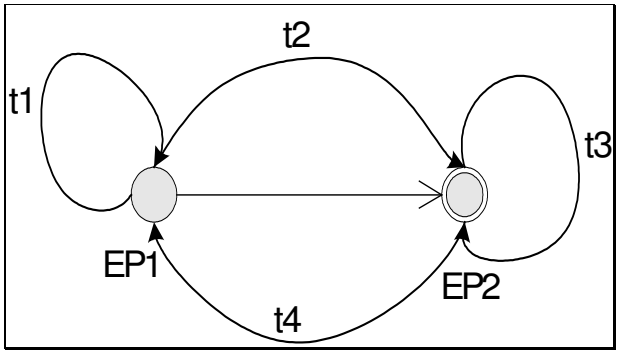

Figure 5.5-1: Point-to-Point Operation Latency Diagram

In the above diagram, the two computing nodes EP1 (End-Point 1) and EP2 (End-Point 2) can be either an AGSI peer or an AGSI server. For a typical communication cycle, the overall time cost or latency can be

$$T = t1 + t2 + t3 + t4$$

- **t1**: the processing time in EP1 (One initializes the communication on EP1 to EP2)
- **t2**: the latency from EP1 to EP2 (network transmission)
- **t3**: the processing time in EP2 (remote objects invocation and execution)
- **t4**: the latency from EP2 to EP1 (Receive a response from EP2 at EP1)

Ideally, the smaller T is, the more efficient of the communication activity is. Computer resources like CPU speed and memory size mainly determine both time t1 and t2. Normally t1/t2 can be reduced when CPU speed gets higher and the available memory size becomes larger. On the other side, both time t2 and t4 are determined by the only network resource, i.e. available network bandwidth.

In order to test the latency, we use clocking function provided in Java to get the machine time before and after certain functions have been executed. With the pair of time clocked, we can get the time difference that represents the consumed time for executing certain functions or operations. Our testing result can be as precise as one millisecond. We measured the latencies of the system in four aspects, which will be discussed in the following sections.

5.5.1 AGSI System Bootstrapping and Session Management

Table 5-1 shows the data that was extracted from the testing result on the general session operation part. The test-bed is introduced and shown in Figure 5.4-1.

AGSI Operations		Time (ms)					Ave. Time (ms)
		1st	2nd	3rd	4th	5th	
AGSI Server Bootstrapping (total)		1813	1937	2000	1828	1890	1890
1	Configuration XML Loading	578	593	578	594	593	587.2
2	Group Advertisement	219	235	234	219	235	228.4
5	Data Manager Creation	500	609	704	593	625	606.2
3	Session Manager Creation	234	188	187	110	109	165.6
4	Security Manager Creation	282	297	297	312	328	303.2
6	Session Scheduler Creation	0	15	0	0	0	3

AGSI Peer Operations							
1	AGSI Proxy creation	1343	1391	1938	1406	1734	1562.4
2	AGSI Agent creation	656	562	312	500	450	496
3	AGSI Server Connection	500	344	641	547	656	537.6
4	Publishing a session	62	46	32	125	47	62.4
5	Receiving a session	62	31	78	16	156	68.6
6	Creating an application	94	80	100	62	90	85.2
7	Binding a session controller	47	187	891	235	969	465.8
8	Disconnecting from AGSI	78	63	62	100	81	76.8

Table 5-1: AGSI System bootstrapping and Session Management Experimentation

The above table lists the latency testing results in three parts, each of which was conducted in five times to avoid system and network environment randomness. One is tested on AGSI server and another on AGSI peer.

We can see it roughly takes less than 2 seconds to start up an AGSI server that is quite acceptable. It also cost about less than 2 seconds for a peer application to load our AGSI proxy that configures itself during the launch time. The connection time to Remote AGSI servers is about half of a second and it has little to do with network speed since the test-bed does not have much network traffic and the remote call incurred by a peer to remote AGSI servers will not take much network bandwidth. Thus it is also true for other types of operations like publishing a session, downloading/retrieving remote sessions and binding of sessions that all take little time to finish.

5.5.2 AGSI Session Publication

Session Publication is a typical activity within AGSI system for advertising sessions so that other peers can discover and consume them. We would want to find out how the performance is when a peer publishes sessions in various amounts of data. In the test, we chose AGSI peer4 to publish sessions onto AGSI server 1. The following Figure shows the latency result. The results were collected from the testing results that ran on

the test-bed as shown in Figure 5.4-1. Below figure shows we tested the latency against the number of session flows. It was done through adding session entries in the Multimedia session configuration XML which is explained in Appendix B.

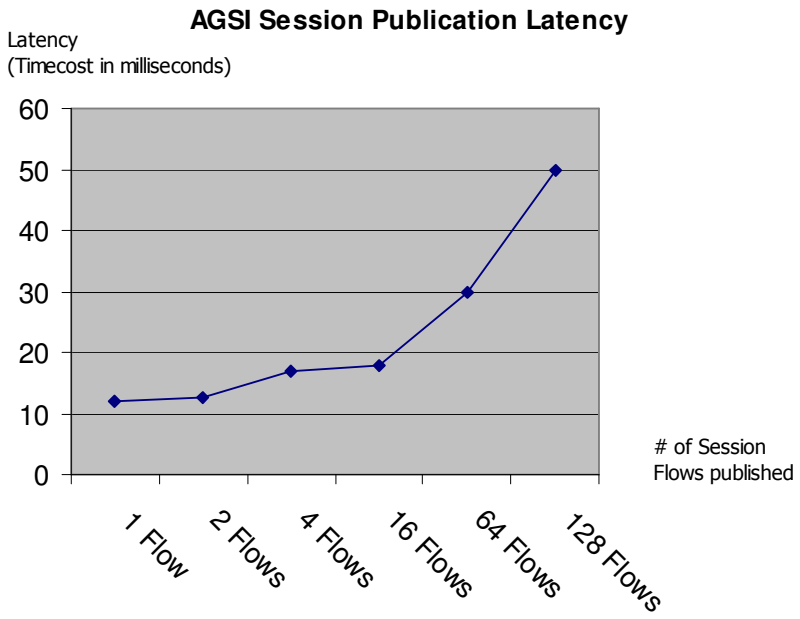

Figure 5.5-2: AGSI Publication Latency

From the above figure, we can see that latency increases as the number of session flows grows. The major reasons for this are slightly due to the network data transferring and largely due to the remote object creation. In another words, the data to be transferred through the network varies little as the size of the data increase. However, the time of creating the related session objects in AGSI system takes a major part of the overall cost.

5.5.3 Session Directory Retrieval

Now we tested the downloading session directory latency in various AGSI environments when there are different numbers of AGSI peers within one AGSI group. Although there are only four AGSI peer end-host in the test-bed, we chose run multiple

AGSI peer client applications in one machine to simulate different peer connections. We again chose three different peers coming from three different sub-networks in our test-bed and tested their latency when downloading AGSI session directory along with the increasing number of AGSI group members. The results were collected from the testing results that ran on the test-bed as shown in Figure 5.4-1.

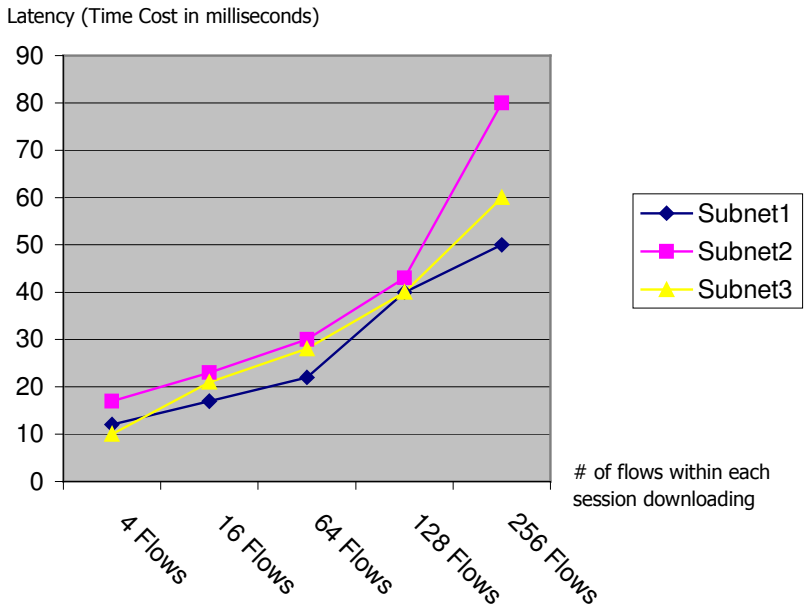

Figure 5.5-3: AGSI Session Directory Retrieval

From the diagram, we can see the latency increases as the traffic of session directory downloading grows. The increases of traffic (due to more group members and higher number of session entries) have resulted in making the session manager to spend more time in retrieving information from customized session directory. As a whole, the three subnets produce very similar results, despite of the lower bandwidth in the wireless network (Subnet 3). However, due to the difference in JINI multicast in different networks, they generated slightly different result of latency.

5.5.4 Group-2-Group Operations

This is a section on the evaluation of the AGSI using the emulated CMS. Under the same test-bed as shown in Figure 5.4-1, the testing results were collected for group-2-group bridging experiments. The detailed network configuration and operational flow is shown in Figure 5.4-3. There are AGSI servers and AGSI peers connecting from two different networks and one common AGSI registry from another network. In between network Net-3 and Net-1, only unicast is supported. In order to send multicast stream from Net-3 to Net-1, an emulated CMS application named as session reflector is installed in both networks. Upon receiving the request for streaming one group session data to the another group that exists in different network, the session reflector will first consult the remote reference of the other session reflect and communicates with it to initiate the unicast session in between. The session reflector from the other network will also establish the connection and start receiving the data and forward to local multicast group to finish the group-2-group bridging.

Finally, to disband the bridging, both the unicast channel and multicast channels have to be disbanded accordingly.

The testing result has been collected in following Table 5-2 and it shows that it takes about one second to finish the group-to-group bridging.

AGSI G2G[10] Operations		Time (ms)					Ave. Time (ms)
		1st	2nd	3rd	4th	5th	
AGSI Group-Group-Bridging		1598	1914	719	576	717	1104.2
1	G2G Initiation	30	40	16	31	16	26.8
2	Local Reflector Creation	60	50	31	16	31	37.6
5	Remote Reflector Creation	70	60	78	15	78	60.2
3	Local Reflector Invocation	628	927	422	359	422	551.4
4	Remote Reflector Invocation	797	837	172	155	170	426.2

Table 5-2: AGSI G2G Operations Experimentation

From the above table, we discovered that overall it takes about 1 to 2 seconds to finish up the whole group-2-group bridging operation. The majority of the time being spent is on the session reflector method invocation. In contrast, the session reflector creation does not take much time. It is understandable that only during the session method invocation, the heavy multimedia objects are initialized and the multimedia sessions are started.

On the whole, the emulated CMS application serves the purpose of demonstrating AGSI group and session capabilities.

[10] G2G: Shorthand for Group-to-group or Group-2-Group

5.6 DISCUSSION AND CONCLUSION

Through the above performance evaluation experiments, we discover that the AGSI system provides relatively good performance result for collaborative group applications. The AGSI infrastructural-level communication cost incurred is very limited and acceptable. AGSI does provide great flexibility and convenience to higher-level applications by playing as a broker for application groups and group peers.

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

With the incorporation of AGSI, OCTOPUS becomes a multimedia-enabled collaborative support infrastructure rather than a simple application tool. AGSI helps manage a host of user objects, application/service objects and session objects in virtual group spaces. The new OCTOPUS can be used to develop a powerful group-based multimedia application system and leave application programmers to work on application specific tasks without putting much extra effort the collaboration infrastructure as well as low level OCTOPUS API invocation which has been largely completed by AGSI.

AGSI also provides the on-the-fly modification feature for both the server and client applications. It helps further reduce the code modification at AGSI middleware layer due to the change of application environment.

AGSI even strives to achieve scalability and extensibility. In the whole picture of AGSI realm, it allows existence of multiple service locating servers, multiple AGSI group servers and multiple AGSI peers as well as various applications provides within the groups. One group can be enabled and hosted by multiple AGSI servers thus the sessions management job can be evenly balanced and the performance can be

guaranteed. Due to the adoption of object-oriented and modularized design nature throughout AGSI design, it is very convenient to plug-in and plug-out new modules into it without affecting much to the rest parts of AGSI. Therefore it is extendable.

6.2 FUTURE WORK

AGSI provides lots of new features and capabilities to OCTOPUS middleware. However there still leaves some room for us to make further improvements.

6.2.1 Provision of a pure P2P computing model

Under current AGSI design, AGSI servers play a central role in governing the whole system and various sessions. It is mostly efficient in collecting member information and consolidating multiple services for collaboration. It requires less communications among different peers and provides stronger sense of group environment for playing as an avatar of some virtual groups.

However everything may come at a price. To enjoy the merits of centralized AGSI servers, we need to deploy them beforehand and set up all the configurations. Sometimes the performance may not be good due to a surge of server workload. Especially when there is no time to set up some independent servers to run AGSI services, a centralized architecture AGSI model will not be applicable any more.

We then would conceive adopting a pure P2P model for AGSI. More specifically, every peer within AGSI framework can even play the role of AGSI server if the peer does want to offer something to the virtual group. A peer with no meaning of sharing something can also play as AGSI server for hosting sessions for other peers. There can be multiple peers hosting AGSI services for the same AGSI group. All the members within the AGSI group are organized and connected in an ad hoc manner. The

workload can be well balanced by those peers who host the AGSI services for the virtual group.

6.2.2 A Unified Identity Management System

Identity management is quite different from group management. Current AGSI design combines them together for simplicity. However if we can separate them and provide a unified identity management for AGSI-enabled applications, it can hail a string of good features. For example, one is Single Sign-On to AGSI world group that allows one peer to access various AGSI groups eliminating the need of establishing its membership repeatedly. A unified identity management also helps reduce storage of membership data and reduce the communications incurred for authentication.

6.2.3 Improvement in Security

Although we have already adopted digital signature technology, it is still unsafe for the famous man-in-the-middle attack where one could claim oneself holding an effectively signed ID without being certified by a trusted third party. To overcome this, digital certificate technology would be indispensable. All remote functional calls will be checked against on the ID information that is conveyed inside a digital certificate. And there shall be some CA (certificate authority) servers set up to issue digital certificates. It shall be also convenient to use a commercial CA server to be its root CA of the whole AGSI group.

6.2.4 Introducing Web Service Technology into AGSI

Web service [21] technology has been around in the IT industry for some time and will continue to be a dominating standard that is used on Internet for exchanging network messages and invoking RPC calls without the hurdle of firewall blocking.

APPENDIX A
CODE SNIPPETS IN INVOKING AGSI API

PART 1: AGSI SERVER INITIALIZATION

```
public class AgsiServer {
    public AgsiServer( ){
          //load AGSI configuration profile XML
           config = new AgsiConfig();

           //Register self with Jini
           reg = new ServiceRegistration(
                 AgsiConfig.getJiniGroups());
           this.register();
           //create db connection
          dbConn = DbConnector.getConnection();


           //remote objects creation
           dataMan     = new AgsiDataManager(dbConn);
           sessionMan  = new AgsiSessionManager(dataMan);
           securityMan = new AgsiSecurityManager();
           //create session scheduler
           scheduler   =
             new AgsiSessionScheduler(dataMan, sessionMan);
           scheduler.start();
    }
    public static void main(String[] args) {
        System.setSecurityManager(new RMISecurityManager());
        server = new AgsiServer();
        System.out.println(">>Server awaits console commands...");
        server.waitForCommand();
    }
}
```

PART 2: AGSI PEER INITIALIZATION

```
public class AgsiPeer {
    //including the proxy within will suffice
    protected AgsiProxy proxy = new AgsiProxy(this);

    public static void main(String[] args) {
     System.setSecurityManager( new RMISecurityManager());

       try {

           AgsiPeer client = new AgsiPeer();

       } catch (Exception e) {

          e.printStackTrace();

      }
}
```

PART 3: CREATION OF SESSION CONTAINER IN AGSI PEER

```
//code snippet in AgsiProxy
Hashtable mmSessionContainerFromLocal = new Hashtable();

//code snippet in AgsiPeer
SessionConfig.constructSessionContainer(
    proxy.mmSessionContainerFromLocal );
```

PART 4: CREATION OF MULTIMEDIA DEVICE IN AGSI PEER

```
//code snippet in AgsiProxy
protected MultimediaDevice mmDevice = null;
mmDevice = new MultimediaDevice(peerConfig.hostGuid);
ServiceRegistration.jiniGroups = peerConfig.jiniGroups;
registerMMDevice();

//code snippet in AgsiPeer
SessionConfig.constructMMDevice(
   proxy.mmSessionContainerFromLocal, proxy.mmDevice );
```

PART 5: PUBLISHING OF MULTIMEDIA APPLICATIONS

```
Enumeration keys = proxy.mmSessionContainerFromLocal.keys();

while( keys.hasMoreElements() ){
     proxy.agents[index].publishApplicationContainer(
       (ApplicationContainer)
        proxy.mmSessionContainerFromLocal.get(keys.nextElement()));

}
```

PART 6: RETRIEVAL OF AGSI SESSIONS DIRECTORY

```
//code snippet within Heartbeat class
Hashtable newMMC =
   agent.agsiSessionMan.retrieveSessionDirectory(
     owner.peerConfig.hostGuid);

if( newMMC != null ){
   agent.mmSessionContainerFromRemote.putAll(newMMC);
}
```

PART 7: SESSION INITIALIZATION AT BOTH SIDES

```
// bind all sessions after publishing…
proxy.agents[index].requestToBindAllSessions(true);

// bind all sessions for consuming…
proxy.agents[index].requestToBindAllSessions(false);
```

PART 8: BRIDGING OF SESSION GROUPS

```
// merge two session groups based on the application spec
public void mergeGroup(ApplicationSpec fromAppSpec, ApplicationSpec
toAppSpec);


// disband two session groups based on the application spec
public void disbandGroup(ApplicationSpec fromAppSpecs,
ApplicationSpec toAppSpecs)
```

APPENDIX B
AGSI CONFIGURATION XML

PART1: AGSI SERVER CONFIGURATION XML

```
<? xml version="1.0" encoding="utf-8”?>
<server>
  <property name="host_guid" value="AGS@32fa2a8d-c7c1-11d7-b55f-
00065b7cca4c" />
  <property name="creator_guid" value="PER@32fa2a8d-c7c1-11d7-b55f-
00065b7cca4c" />
  <property name="host_ip" value="172.18.178.85" />
  <property name="host_name" value="socf-cir-010" />
  <property name="jini_group" value="AGSI_JINI" />
  <dbconn>
    <property name="driver" value="sun.jdbc.odbc.JdbcOdbcDriver" />
    <property name="url" value="jdbc:odbc:AGSI_SQL" />
    <property name="UserID" value="richard" />
    <property name="password" value="password" />
  </dbconn>
</server>
```

PART2: AGSI PEER CONFIGURATION XML

```
<? xml version="1.0" encoding="utf-8”?>
<peer>
  <property name="host_guid" value="PER@0de33fe1-b698-11d7-a4a1-
00065b7cca4c " />
  <property name="host_ip" value="172.18.178.85" />
  <property name="host_name" value="socf-cir-010" />
  <property name="jini_group" value="AGSI_JINI" />
  <property name="agsi_guid" value="AGS@32fa2a8d-c7c1-11d7-b55f-
00065b7cca4c" />
  <property name="agsi_guid" value="AGS@d5d3b86e-b537-11d7-8588-
0050da7912f0" />
  <property name="public_key_file" value="null" />
  <property name="private_key_file" value="null" />
  <property name="password" value="null" />
  <property name="peer_name" value="Richard" />
  <property name="peer_email" value="null" />
  <property name="peer_address" value="null" />
  <property name="peer_organization" value="null" />
  <property name="peer_contacts" value="null" />
  <property name="peer_weburl" value="null" />
  <property name="peer_description" value="null" />
</peer>
```

PART3: AGSI MULTIMEDIA DEVICE CONFIGURATION XML

```
<? xml version="1.0" encoding="utf-8”?>
<mm_device>
 <session name="Java RMI Tutorial">
    <acl>
       <oid type="agsi" value="AGS@5eccc036-b44c-11d7-81c2-
00065b7cca4c" allowed="yes" />
       <oid type="group" value="GRP@1cf2fe3a-b673-11d7-b51e-
00065b7cca4c" allowed="yes" />
       <oid type="role" value="ROL@5eccc036-b44c-11d7-81c2-
00065b7cca4c" allowed="yes" />
       <oid type="member" value="PER@5eccc036-b44c-11d7-81c2-
00065b7cca4c" allowed="no" />
     </acl>
     <stream name="AVStream">
        <flow name="MusicFlow" direction="PRODUCER"
device_name="AudioFileSource"  device_locator="file:chen.wav" />
        <flow name="VideoFlow" direction="PRODUCER"
device_name="Camera" device_locator="vfw://0" />
     </stream>
     <stream name="VoiceStream">
       <flow name="VoiceFlow" direction="PRODUCER"
device_name="AudioFileSource"  device_locator="file:out.wav" />
     </stream>
   </session>
</mm_device>
```

REFERENCES

[1] C.H. Zhang, T.K. Chin, K.Y. Koh, G.M. Ong, C.H. Peng, H.K. Pung, and S. Suthon, "OCTOPUS: A middleware for multimedia communication," IMSA, August 2002.

[2] C.H. Zhang, "The design and implementation of a Jini/Java based AV Streaming Framework", Master's thesis, School of Computing, National University of Singapore, October 2002.

[3] Suthon Sae-Whong, "QMan an adaptive end-to-end quality of service management in OCTOPUS" Masters Thesis, School of Computing, National University of Singapore, May 2003.

[4] J.K. Yao, "The design and implementation of a service locating manager in OCTOPUS: a multimedia communication middleware", Master's thesis, School of Computing, National University of Singapore, October 2002.

[5] Ong Geok Meng, "A Dynamic Protocol Framework in OCTOPUS: A Multimedia Communication Middleware", Honors Year Project Report, 2002

[6] Robin, "Groups Merging and Disbanding in the Internet", Master Thesis, School of Computing, National University of Singapore, 2003

[7] Object Management Group (1996) Control and Management of A/V Streams Request for Proposal. OMG Document: telecom/96-08-01

[8] CSCW (Computer-Supported Cooperative Work). CSCW Bibliography. CSCW pages @ Technische Univ. Muenchen, Germany. CSCW bibliography, CSCW home-CSCW bibliography-CSCW links. This is a gateway to the bibliographic database on CSCW and related topics maintained at Applied Informatics, 1/10/99. http://www.telekooperation.de/cscw/cscw-biblio.html

[9] Reinhard, Schweitzer et al, CSCW tools: Concepts and architectures , 1994

[10] Clarence A. Ellis: Groupware: Overview and Perspectives. Wissensbasierte System 1991: 18-29

[11] W. Keith Edwards, "Session Management for Collaborative Applications", ACM SIGGROUP, 1994

[12] Gabriel Dermler, et al, "JVTOS Multimedia Telecooperation Interconnecting Heterogeneous Platforms", 1994

[13] Erik Wilde, Christoph Burkhardt, "Modelling Groups for Group Communications", 1995

[14] Erik Wilde, Bernhard Plattner. "Transport-Independent Group and Session Management for Group Communications Platforms", 1997

[15] M.Handley, J.Crowcroft, and C.Bormann. The Internet Multimedia Conferenceing Architecture. Internet Draft, MMUSIC Working Group, February 1996.

[16] Casner, "Frequently Asked Questions (FAQ) on the Multicast Backbone", May 6, 1993, ftp://venera.isi.edu/mbone/faq.txt

[17] Mark Handley and Van Jacobson. SDP: Session Description Protocol. Internet draft, MMUSIC Working Group, November 1996 (ftp://ds.internic.net/internet-drafts/draft-ietf-mmusic-sdp.00.{txt, ps})
[18] M. Handley, C. Perkins, E. Whelan. SAP: Session Announcement Protocol. Internet Draft, MMUSIC Working Group, December 1999.
[19] Mark Handley, Henning Schulzrinne, and Eve Schooler. SIP: Session Initiation Protocol. Internet Draft, MMUSIC Working Group, December 1996.
[20] Sherlia Shi, Design of Ovelay Networks for Internet Multicast (2002). Ph.D. dissertation. 2002, Washington University in St. Louis
[21] Web service: a programmatic interfaces used mainly on the internet http://www.w3.org/2002/ws
[22] IGMP: Internet Group Management Protocol http://www.networksorcery.com/enp/protocol/igmp.htm
[23] Sun Microsystems, Inc. "JINI™ Architectural Overview" Technical white paper, 1999.
[24] Li Gong. A secure identity-based capability system. In Proceedings of the IEEE Symposium on Security and Privacy, pages 56--63, Oakland, California, U.S.A., May 1989.
[25] Object Management Group (OMG), "Control and management of Audio/Video Streams", CORBATelecoms: Telecomm Domain Spec version 1.0, Group Management Protocol, Version 2", RFC 2236, and November 1997.