NOTIFICATION SERVICES IN A DISTRIBUTED CONFERENCING SYSTEM

By

SWATI PATANJALI SHUKLA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2000

Copyright 2000

by

Swati Patanjali Shukla

I dedicate this work to my father, mother, sister and grandmother.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere thanks to Dr. Richard Newman, my thesis advisor, for introducing me to this thesis topic and for his constant guidance and support throughout my entire stay at the University of Florida.

I thank Dr. Douglas Dankel and Dr. Haniph Latchman for being so patient and accommodating with me.

The DCS group deserves my sincere appreciation for their support and critical observations, making this a wonderful project to work on.

My friends deserve my sincere thanks for their encouragement throughout my studies.

I can never thank enough to my sister and grandmother for being with me and believing in me always.

Last but not least, I would like to thank my mother and father for their infinite love and support, and for their constant guidance and encouragement. It is to them I dedicate this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

NOTIFICATION SERVICES IN A
DISTRIBUTED CONFERENCING SYSTEM

By

Swati Patanjali Shukla

December 2000

Chairman: Dr. Richard Newman
Major Department: Computer and Information Sciences and Engineering

The development of groupware applications dedicated to support different group work activities has brought about new challenges to understand better group interaction and how technology can affect and enhance it. Notification mechanisms have been used to improve the user's knowledge of the actions taking place in the group that may affect his own work. However, while most of the groupware applications developed use notification mechanisms to support synchronous interaction among users, they have rarely been used to support asynchronous interactions.

This work presents the notification system developed for the University of Florida Distributed Conferencing System Version 2, DCS v.2. The notification system developed is a flexible and friendly system that provides a set of useful services to allow users to be informed about what is happening in the collaborative environment, even if they are not actively participating. It also takes into consideration the role each user plays in the group. The users can select the events that interest them and the manner in which they

want to be notified about those events (how and when) through Graphical User Interface (GUI). Moreover, the system provides Application Programmers Interface (API) that allows programmers to use it as a tool to communicate to the users from within DCS v.2 services and applications.

CHAPTER 1
INTRODUCTION

The concept of notification is widely used in everyday systems. Mechanical plants, chemical plants, cars, elevators and computer systems all use notification mechanisms in different forms. The present work concentrates on notification mechanisms for computer systems.

Notification mechanisms are used to inform agents about different events happening in the system; these are occurrences of observable (in a program sense) activities. Messages, page faults, hardware traps, time expiration, control flow of the program, etc. are events.

The process of notifying an agent of an event involves three steps.

1. **Raise or signal the event.** In this step, predefined events are raised by the operating system, called *system events*, or raised by an application program, called *user events*. For example, division by zero causes the operating system to raise a system event; reaching a given value in a control application can cause the control application to raise a user event.
2. **Event handling.** In this step, the information about the event is sent to the event handler, which in turn may take some actions. Conversion of the received value or creation of the message to be sent to a user is examples of event handling.
3. **Event notification.** In this step, the event handler sends the final event information to the selected set of recipients.

1.1 Computer-Supported Collaborative Work(CSCW) and Groupware

Much of a person's work happens in a group rather than in an individual context, which makes group support an important issue. CSCW is a multidisciplinary field that studies group work and how the computer and related technology affect group dynamics. This new area of research gathers together researchers from a variety of specializations: computer science, cognitive science, psychology, sociology, anthropology, ethnography, management and management information systems [1], each of them providing different methodology and perspective to study groups and how to support them.

On the other hand, groupware is the software that facilitates group work by assisting members of the group in communicating, collaborating and coordinating their activities. Ellis et al. [2] defined 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. (40)" Examples of groupware are electronic mail, bulletin boards, desktop conferencing, video conferencing, collaborative authoring tools, voice applications, decision support systems, group calendars and workflow systems among others.

Communication, collaboration and coordination are three basic areas for groupware development. Communication is "the imparting or interchange of thoughts, opinions, or information by speech, writing, or signs (298) [3]." Communication is very important to support group work because a group's members must exchange information. Communication is the underlying technique that supports the transmission of work for further processing or use, as well as the transmission of information about the progress of the group activities to coordinate them.

Coordination is “the combination in suitable relation for most effective or harmonious results (502) [4].” Coordination is an activity required when groups of people work together, since group work is based on the organization of the activities of the members of the group. Action performed by each member of the group must be coordinated so that conflicts can be avoided. Generally, access to shared objects is coordinated to assure the object’s integrity. Moreover, coordination techniques are used as part of a process model to identify *who* does *what* and *when*, as well as to identify where the user is within the process. Therefore, coordination is a technique used to support collaboration; coordination does not require the actual transmission of work, but instead requires information on how the processes advance.

Collaboration is the act of working together; it is groupware’s primary goal. Collaboration requires that people share information (communicate and coordinate). Ellis et al. [2] affirmed that “what is needed are shared environments that unobtrusively offer up-to-date group context and explicit notification of each user’s actions when appropriate. (40)” Collaboration needs either implicit or explicit communication and coordination. Explicit communication is needed when it is important to identify the different activities the members should perform, while implicit communication is needed when the activities of each member are determined by the role played by the user in the group. Moreover, communication and coordination activities may be performed before or during the period of work (or both). For example, in the process of writing a document each member of the group may be responsible for a part of the document, requiring common planning to assign the different sections to the users prior to the creation of the document. In contrast, synchronous editing of documents requires on-line communication

of the changes each member performs on the document and synchronous coordination of the members' activities to assure that no more than one member modifies a single section of the document at a time.

Notification mechanisms are used to aid in the coordination of group work. Notification is communication specifically supporting coordination: knowing when a part of a process has occurred, for example when a document is modified, or a user joins the group. It can be used to allow users to specify what they want to know about and when, without producing an explicit process model.

1.2 Notification in Groupware Applications

In a single user environment it is important to notify users when constraints are violated, or when automatic operations provoke triggers or alerts. Notification is even more vital in a multi-user environment, particularly in collaborative environments, because users must know when other users make changes that affect their work.

Dourish and Bellotti [5] define awareness as the "understanding of the activities of others, that provides a context for your own activity (107)" and affirm that awareness of others' activities is critical for successful collaboration in a group work. Commonly, awareness is supported in CSCW by active information generation mechanisms separate from the shared workspace. Notification is a mechanism used to support awareness.

1.3 Motivation

Most groupware applications use notification mechanisms to support synchronous interaction between users of a group, but they have rarely been used to support asynchronous interactions. For example, in software that supports synchronous edition of documents, a user is aware of the modifications made in the document while he is

actively participating in the writing session, but once he leaves the active session there is no automatic mechanism to inform him that the document has been modified. Similarly, in conferencing system a user that participates in a conference is aware of the actions taking place in the conference while he is actively participating in the session, but once he leaves the active session, although the conference continues, the user is not aware of the actions taking place, some of which may interest him. Notification mechanisms that reach the user even though he is not participating in the active session can help the user to know about those important actions that affect him.

This thesis concentrates on notification services for one of the groupware applications developed for conferencing support, the University of Florida Distributed Conferencing System version 2 (DCS v.2).

1.4 Distributed Conferencing System (DCS)

DCS is an example of a distributed system designed to support conferencing over a wide area network (WAN). This system allows geographically separate users to collaborate in the preparation of documents, graphics, software tools, as well as make demonstrations. The first version of DCS, referred as DCS v.1, supports synchronous notification of some of the events going on in the conference through the status window (this is a window that always exists while the user is running DCS and allows him to access information about the conference). However, DCS v.1 offers limited asynchronous notification services: the system notifies users when they have been invited to join the conference, and to request their votes [6]. Both synchronous and asynchronous notification mechanisms are hard-wired in the system, giving no flexibility to the users to indicate if they require them or not.

Experience with DCS v.1 revealed a need for more sophisticated notification services, especially to inform members not actively participating in the conference about actions taking place that may interest them. Also, a notification service may be used to inform appropriate users of the presence of problems with the system (for example, if a server is not available and human intervention is required).

We developed a notification system for DCS v.2 that keeps users informed of actions going on in the conference, even if they are not logged in the computer. The notification system is flexible and allows users to select the event or events that interest them, the way they want to receive the information about the event (more than one method of notification is provided) and when they want to be notified of those events. Security is also taken into consideration by controlling the user access to information related to events based on the user's role (set of privileges and responsibilities attributed to a person).

Although the notification system was developed for a specific system, DCS v.2, the techniques and concepts managed can be used and applied to other areas.

1.5 Organization of Thesis

Chapter 2 contains a survey of the systems relevant to this thesis work. Chapter 3 captures the complete requirements. Chapter 4 discusses the design and relevant issues in detail. Chapter 5 explains the relevant implementation details. The last chapter presents conclusions, future enhancements and modifications.

CHAPTER 2
WORK RELATED TO NOTIFICATION SERVICES IN COMPUTER SYSTEMS

This chapter presents some related work and explains the motivation for notification services. Section 2.1 gives a brief introduction to CSCW and groupware. Sections 2.2 through 2.6 describe some of the groupware applications developed. Section 2.7 describes network notification services. Section 2.8 describes some of the notification mechanisms developed in Unix. Section 2.9 talks about Unix delivery mechanisms. Section 2.9 gives a brief introduction to active databases and presents two developed active databases. Finally the last section summarizes and discusses the motivation for notification services.

2.1 CSCW and Groupware

Today, we observe that computers and networks are beginning to form social spaces where people present themselves, meet with other people, exchange information, play games together, do business or jointly look for information. CSCW or "Computer-Supported Cooperative Work" is the study of how people work together using computer technology. Typical applications of CSCW include email, awareness and notification systems, videoconferencing, chat systems, multi-player games and real time shared applications (such as collaborative writing or drawing).

CSCW Domain

CSCW includes many computer science notions and technologies including networks, multimedia, object oriented concepts, virtual reality and artificial intelligence.

The most appropriate technology of groupware applications is used for specific cooperative communication tasks.

CSCW Dimensions

There are two dimensions that make up the CSCW domain:

1. Time

Real Time - This is when communication occurs at the same time.

Asynchronous Time - This is when communication occurs at different times.

2. Place

Same Place - This is where people meet in the same room.

Different Place - This is where a meeting occurs where the participants are in geographically distributed locations.

Groupware is technology designed to facilitate the work of groups. This technology may be used to communicate, cooperate, coordinate, solve problems, compete or negotiate. The term groupware is ordinarily used to refer to a specific class of technologies relying on modern computer networks, such as email, newsgroups, videophones or chat.

Groupware Applications

Groupware has synchronous and asynchronous applications.

1. Asynchronous Groupware

Email is the most common groupware application. .

Newsgroups and mailing lists are similar in spirit to email systems except that they are intended for messages among large groups of people instead of one-to-one communication. In practice the main difference between newsgroups and mailing lists is

that newsgroups only show messages to a user when they are explicitly requested (an "on-demand" service), while mailing lists deliver messages as they become available (an "interrupt-driven" interface).

Workflow systems allow documents to be routed through organizations through a relatively fixed process.

Hypertext is a system for linking text documents to each other. Whenever multiple people author and link documents, the system becomes group work, constantly evolving and responding to others' work. Some hypertext systems include capabilities for seeing who else has visited a certain page or link, or at least seeing how often a link has been followed, thus giving users a basic awareness of what other people are doing in the system.

Group calendars allow scheduling, project management and coordination among many people and may provide support for scheduling equipment as well. Typical features detect when schedules conflict or find meeting times that will work for everyone. Group calendars also help to locate people. Privacy, completeness and accuracy are typical concerns.

Collaborative writing systems may provide both real-time support and non-real-time support.

Word processors may provide asynchronous support by showing authorship and by allowing users to track changes and make annotations to documents. Authors collaborating on a document may also be given tools to help plan and coordinate the authoring process, such as methods for locking parts of the document or linking separately-authored documents.

2. Synchronous or Real-time Groupware

Video communications systems allow two-way or multi-way calling with live video, essentially a telephone system with an additional visual component.

Talk systems permit many people to write messages in real-time in a public space. As each person submits a message, it appears at the bottom of a scrolling screen. Having listing chat rooms by name, location, number of people, topic of discussion etc. usually forms talk groups.

Decision support systems are designed to facilitate groups in decision-making. They provide tools for brainstorming, critiquing ideas, putting weights and probabilities on events and alternatives and voting. Such systems enable presumably more rational and even-handed decisions.

2.2 Conferencing Systems

Conferencing Systems provide computer-based facilities for the support of conference activities. These systems are focused on providing a set of tools to coordinate the interaction among people and to support activities carried out during a conference. For example, conversations, manipulation and creation of documents and demonstrations are some of the activities that can be supported.

Some of the developed conferencing systems only support synchronous interaction among people located in the same room, for example Cognoter [7]. Other systems provide support for synchronous or asynchronous interaction among people dispersed over different locations, for example Mermaid [8] and DCS v.1 (Distributed Conferencing System Version 1) [6].

2.2.1 Distributed Multiparty Desktop Conferencing System: MERMAID

Mermaid [8] (Multimedia Environment for Remote Multiple Attendee Interactive Decision making) is a system that allows the interchange of information using video, video and multimedia documents. The purpose of Mermaid is to provide computer and communication tools that allow people geographically separated to work together, at the same or different time, in an effective and efficient way. Mermaid can support any number of participants and the number of conferences that it can manage is dictated by the processing speed of the workstation and the communication media.

Mermaid is implemented using a client-server architecture with specialized servers including:

1. A Conference Management Server (CMS) that manages the progress of the conferences.
2. A Conference Information Server (CIS) that provides information about the conferences (e.g. membership, agenda, and advance notices).
3. A Document File Server (DFS) that stores and retrieves the conference's documents.
4. A Local Communication Server (LCS) that manages the communications within the local area network (LAN).
5. A Multi-Domain Communication Server (MCS) that manages the communications across domains over a Wide Area Network (WAN) and
6. Clients (C) that provide to the members of the conference a friendly user interface to interact with the servers and other clients.

Mermaid users interact with the system and other users through a set of windows: the conference window (where the user can access information about conferences, join a

conference, etc.), the shared window (where documents are shared among all the participants), personal window (the objects displayed in this window are not seen by others), video window (display participants' video images) and the status window (shows the picture of the participants and the conference's status).

Mermaid offers support for activities performed before, during and after conferences.

1. During the conference, the user interacts with the rest of the members through the windows supported by the system, as well as by a voice channel, which is available to enhance the communication.
2. Before and after the conference, Mermaid users have tools to manipulate multimedia documents. Additionally, Mermaid offers mechanisms for notifying people about the initiation of the conference using the *convener*. The convener can summon participants to a conference in a two ways. First, the convener chooses a conference notice (which contains participants names) from the notice list and notifies those participants. Second, the convener selects individual names from a directory and notifies them.

Mermaid supports synchronous notification through the shared and status windows. The shared window notifies all active members about the actions performed by any user, and the status window informs all active members of the active members in the conference. However, Mermaid does not offer notification services to those users that are not currently participating in the conference. The only asynchronous notification supported by Mermaid is the convener that notifies users about the initiation of a new conference.

2.2.2 Distributed Conference System, Version 1. DCS v.1

DCS v.1 is a distributed system that provides real-time support for collaborative work. It offers a set of shared applications that allow groups of users to collaborate in the preparation and demonstration of text, figures and software [6].

DCS's architecture is based on cooperative distributed processes:

1. The Central Conference Server (CCS), which always exists and provides access and coordination between conferences, does error recovery and monitors the performance of Conference Managers (CMs).
2. The Conference Manager (CM), one per active conference, is responsible for supplying the discussion window and basic conference control.
3. The User Manager (UM), which acts as a dialog manager for the individual user active in a conference.
4. The Application Manager (AM), which is in charge of the communication between the application and the CM and interacts with the Application Window (AW). Each application has its own (logical) AM.

The user interacts with the system through different windows:

1. The Status Window (SW) allows users to access information about conferences (i.e., make queries about existing conferences) and conference control (i.e., make motions, voting). This window always exists while the user is running DCS.
2. The Discussion Window (DW) is opened by default when the user joints a conference. Through this window the user establishes real-time (synchronous) communication with the other conference participants.
3. The Execute Window (EW) is opened by the user and allows him to access the shell. This window can be exported to other users, allowing them to see the

execution in the source window. This is the only window that requires the user to gain control of it before accessing it. The rest of the execute windows are synchronized to the master using the What You See Is What I See (WYSIWIS) paradigm.

4. The Text Edit Window allows a user to interact and use DCS's text editor, Mace [9].
5. The Graphics Edit Window allows users to interact and use DCS's graphics editor, Ensemble [10].

DCS v.1 offers support for both extra-conference and intra-conference activities. The user can ask for a list of conferences, can initiate a conference or request to join a conference. If the user is a member of a conference he can make queries, motions and vote.

DCS v.1 allows users to be aware of the actions taking place in the conference while they are actively participating, but no notification is provided to those users that are not actively participating. The only asynchronous notification services supported by DCS v.1 are the invitation to join a conference and the request for the user's vote.

2.3 Multiuser Text Editors

Multiuser text editors allow users to compose and edit a document jointly. They can be classified based on the time restrictions placed for the collaboration process as shown:

1. Non-real time or asynchronous text editors, that do not support simultaneous edition of documents for more than one user. In these cases collaboration is

achieved by using annotation mechanisms among the people working on a document edition as in the case of Quilt [11].

2. Real time or synchronous group editors, which allow a group of people to edit the same document at the same time. These systems provide different mechanisms to control users' access to the document. Generally, only one user can modify part of the document and the rest can see the changes or modify other parts.

2.3.1 Collaborative Document Production Using Quilt

Quilt is a tool developed to provide support for asynchronous collaboration in document creation [11].

The process of collaboration supported by Quilt includes passing on-line drafts back and forth among collaborators. Each collaborator has access rights to the document depending on his role (full access for reading and writing, only reading, only annotation access, etc.). The roles are defined and customized by the authors of the document. The communication among authors is established through annotation mechanisms that support both text and voice annotations. Those mechanisms include four types of annotation: comments, revision, private note and directed message (in this case an e-mail message is sent). Quilt also has mechanisms to aid in the coordination and activity planning: triggering, logging and notification.

The user can set up triggers based on specific actions. For example, a person can ask to be notified when substantial changes have been made to the document or to be notified when the other user finishes dealing with the comment. Notification mechanisms supported by Quilt are integrated with e-mail. Quilt keeps a log with the changes performed to the document, keeping track of the date and time.

2.3.2 MACE

Mace is the DCS v.1's text editor, but also can be run as a stand-alone tool. It supports real-time fine-grained (character level) concurrent editing [9]. It is a modal editor that has three modes: scroll, edit and update.

Scroll: When the editor is loaded it is in scroll mode. In this mode the user can see any part of the file by scrolling it. The users can also choose to see the locks placed on the file and by clicking on the locks, see information about who has the lock and request to see the changes. The changes can be seen as single updates (to see the changes the other user has made until that moment and come back to scroll mode), and real time updates (this option synchronizes the viewer's screen to the screen of the remote editor, and the user will see the changes as they occur).

It is important to highlight that the user that locks a section for modification also defines the access for others to see his changes. If he does not want others to see his changes, he can select view locks, preventing others from seeing the changes until he finishes.

Edit: The user enters this mode when he acquires a pair of edit locks that delimit the region of the file that he wants to modify. More than one user can modify the same file by requesting locks for different sections. The user can only modify one section (delimited by the pair of locks) per window at a time; if he wishes to modify more than one section, it is necessary for him to open more than one edit window for the same file. A user can place view locks on the locked section that prevent other users from seeing changes until the view locks are released or the edit locks are released. The view locks are implicitly released when the edit locks are released.

Update: This mode occurs when the user, having finished making the changes, releases the edit locks. The user may then abort or commit the changes to the original file. All other users will be able to see the changes to the section.

There are three modes for viewing a text file: static allows portions of a file to be viewed as they were before modifications began; snapshot captures a snapshot of a current state of a locked portion of a file; and, synchronous which allows the viewers to see changes as they occur.

Mace provides synchronous notification mechanisms only. A user can see the changes made in the document as they occur while participating in the active session that the document has changed.

2.3.3 Grove

Grove (Group Outline Viewing Editor) is a text editor designed to support synchronous edition of documents by a group of people [2].

In a Grove *session* each user can manipulate one or more *views* of the text being edited in multiple overlapping windows. A *view* is a subset of the items in the document determined by read access privileges. Views can be private (only a particular user can read), sharable (can be read by a set of users) or public (readable to all users).

Grove's users see the outline of the document and by clicking in the outline they can see the underlying text. If a user is not allowed to see parts of the text, a gray box besides the text indicates it.

The group windows of Grove display information about the views, as well as who is able to use the window and who is actually participating in the session at a given time. This information is provided by displaying images of the people who are members of the view (or only their names if no image is available) along the bottom of the window. Thus,

as users join and leave the session their images or names appear and disappear from the window.

Grove only supports synchronous interaction among group's members. It notifies active users of the changes being made in the document and of changes in the participants of the session. Grove supports fine-grained (keystroke level) concurrent editing and notification. Thus, members can see the changes being made by other users as they occur. However, Grove does not provide notification to users that are not in the session.

2.4 Workflow

The workflow concept has evolved from the notion of process in manufacturing and the office [12]. Such processes have existed since the industrial revolution and are products of a search to increase efficiency by concentrating on the routine aspects of work tasks.

A workflow consists of activities involving the coordinated execution of multiple tasks performed by different processing entities and that achieve a common goal within an organizational frame.

A Workflow program is an intelligent system that encompasses variables and rules about the work being performed and the people performing it. It can balance workloads, notify users of lapsed or time-dependent tasks and ensure that interdependent jobs progress in parallel [13].

These systems usually contain logic about how to change the route of information flow depending upon what occurs at every workstation. Good workflow programs also allow work originators and systems supervisors to keep track of individual jobs and determine their status.

2.4.1 InConcert

InConcert [14] is an object-oriented client-server workflow management system. As a document routing type of workflow technology, it is based on the idea that since most of the corporate information is contained in documents, a workflow should deliver the right documents to the right people at the right time, streamlining work, liberating users from the routine parts of processes and, thus, allowing for more productive time.

Sarin et al. [15] studied the problem of modeling and managing the activities of a collaborative process and evaluated it from two points of view:

1. The organization that must define and manage the set of activities and
2. The individuals, who need to know their role in the overall process to translate it into the vocabulary of the computer system.

The requirements for a system that addresses both the above organizational objectives, as well as improves individuals' abilities to manage their work are as follows:

1. Permit an explicit representation of the collaborative process. The system should provide tools and support for modeling the structure of the process and the roles of the individuals participating in the process and defining what documents and applications tools should be used to perform each activity.
2. Route the work to individuals based on the role they play on the process. The system should give support to the user by indicating him that new work must be done, what actions should be done and what application tools may be helpful.
3. Allow human judgment in defining when a task has been finished, and what the sequence of tasks might be.
4. Be flexible, and allow *"mutable processes"* to deal with exceptions.

5. Allow users to monitor the processes, by knowing the status of the tasks, what tasks has been completed and so on.
6. The system should operate in a distributed, heterogeneous, network and support different applications and system services on the organization.

The primary concept managed by InConcert is that of a *job*, which is a multi-person collaborative activity. A job is consists of *tasks*, which are units of work that have ordering dependencies among them. Each task is assigned to a *role*, which represents a user or program that will perform the task. Each task (and job) also defines the documents and applications associated with the task (or job) and this constitutes its workspace or context.

Two kinds of user interfaces are provided by InConcert, the interface for the workers, oriented towards accessing and performing tasks and the administrative interface oriented towards process definition and system administration.

InConcert maintains a record of significant events in the *audit log*. Sarin et al. define an event as “the execution of an operation on some object (220) [15]”; they permit actions to be associated with document checkin and checkout events, tasks status changes and job creation or completion events. Examples of events are tasks becoming ready to work on or overdue, document checkin and checkout and events define by the application.

InConcert notifies users about pending jobs and when tasks become ready. Also, InConcert’s users have facilities to select events of interest. Users can place themselves in “subscriptions lists” associated with documents to receive notification when the documents are modified or can define triggers to invoke specific actions when certain

events occur. The action invoked can be the creation of a new InConcert job, notification of users via e-mail or invocation of an application-supplied procedure via remote procedure call.

2.5 Session Management for Collaborative Applications

Edwards [16] defined the Session Management as "the process of starting, stopping, joining, leaving and browsing collaborative situations across the network (323)." He states that application developers implement subsystems to perform session management on a per application basis, thus limiting cooperation among applications and code reuse and offering few session management facilities: they are not very robust, flexible or powerful. Therefore, it is very difficult to build session management systems, which permit the easy flow of information among applications following these approaches.

Current models of session management need the user to take actions to perform some of the coordination activities required to begin a collaborative session. *Initiator-based* session management systems require the user who starts the collaborative session to notify others about it; *Joiner-based* session management system requires the user who joins the conference to browse or know a priori that the collaborative session will take place. Edwards classifies these approaches as *explicit session management*.

Edwards' approach is to use the act of opening the object of the collaboration as the potential for collaborative activity. Therefore, the system would "detect the potential for collaboration inherent in the fact that multiple users are working in the same object, without the need of naming sessions or browsing lists of sessions to accomplish

rendezvous (325) [16].” This approach is classified as *implicit session management* and it is *artifact-based*.

Activity information is used as the basis to provide a powerful session management system. It maintains information about the tasks being performed across the network: the users on the systems, the tasks they are performing and the objects of those tasks.

An application must publish activity information to participate in the implicit session management. The session management service uses the information published by the applications to find potential collaborative situations (i.e., two activity tuples that contain the same object token) and takes the “appropriate” action (specified by the user or the application).

Intermezzo is the implicit session management system implemented by Edwards. It provides a set of programming libraries, run-time services and conventions that applications can use to participate in the session management service. Intermezzo is implemented using a client-server model. To make information available across the network, Intermezzo uses a publish/subscribe model; a database is used to save the information published and to make it available to other applications.

Applications that participate in the session management system publish an ActivityRecord resource that represents themselves, their user and the data they are accessing, when they start. Applications can also request and handle notification from the server when a potential collaboration exists, as well as to notify the server about any desired user or application policy constraint. Applications can access and update the information stored as long as they have appropriate permissions.

When the Intermezzo server receives a new ActivityRecord, it searches the database for overlaps in the object attribute. When a confluence occurs, the server retrieves an attribute (Colab Action) that indicates the action that has to be executed, for example, generate an event to the application or run a program. It is in the application developers' hands to decide the different actions the applications will manage. Even though an application may receive an event from Intermezzo notifying it that a potential collaboration exist, it may chose to ignore it.

To summarize, Intermezzo provides a set of facilities that can be reused by the application programmers, freeing them from the task of building session management facilities on a per conference basis; it provides for changes in the session management behavior across applications; and it generally provides greater support than the session management facilities provided by most current applications.

2.6 General Systems

2.6.1 ConversationBuilder

Conversation Builder is a system developed to support collaborative processes. These collaborative processes are viewed as sequences of actions in which the actions of one individual can affect the possibilities for actions of other participants in the process [17].

A study of collaborative processes by Kaplan et al. [17] revealed that work activities have a highly situated nature. This characteristic raises the problem of how to support work when the tasks that need to be supported can change continuously. A tool to support this kind of work must be flexible to deal with changes and also it should offer an

active support environment to aid users in varying degrees, depending on the task on hand.

Conversation Builder deals with this problem through the mechanism of *protocols*, which allow different activity types and policies to be defined and through the mechanism of obligations, which are used dynamically to weave individuals' activities together. Kaplan et al. define protocols as "context-sensitive specifications of the actions that can be performed in an activity and their effects (380) [17]."

Conversation Builder architecture follows a model that defines each "activity" in which the user is engaged as a "conversation" in which the user "makes utterances" or "perform actions." Additionally, the performance of an action modifies the space for actions for other participants.

Conversation Builder aids users to a degree by enabling them to be aware of the activities in which they are engaged, the relations among those activities, the legal actions that can be performed in each activity, the hypertext structure and the relevant actions of others.

Conversation Builder supports a notion of awareness of other users because whenever one participant in a conversation performs an action, the other participants are immediately notified. The system can also display information about who else is looking at particular document, who is editing what, which participants are actively working on it and so on.

2.6.2 Active Mail-A Framework for Implementing Groupware

Active Mail is a framework developed to implement groupware applications that interact with the user in a more efficient way. The applications implemented in Active

Mail use non-intrusive mechanisms to ask users to join the application, tolerate delayed responses and are easy to use because of their similarity with e-mail [18].

Active Mail made an extension to mail by adding *active messages* that are piggybacked onto an ordinary mail message. Active messages are entities that contain communication ports, which provide communication and cooperation facilities to interact with other users. The interaction is established between the sender and the receivers of the message and to those users that receive a copy of the message. Also, interaction can be established with applications connected to a communication port in the message. An active message appears as an application window in the user's screen.

Active Mail consists of agents, users and applications and bi-directional communication channels. Agents can perform different actions, for example an agent with a given set of ports can change its environment by creating a new application agent, send a copy of its ports to a user or discard its ports. Also, an agent can send or receive messages through any of its ports.

Similarly, an application can send to a user a copy of its port to another application, allowing the user to interact with that new application. Users are provided with interfaces to the system that allow them to create application agents and to interact with their applications' ports.

An important characteristic of the active messages is that they can *change*. For example, they change when a new contribution to a conversation arrives. Information about these changes is sent to the users via notification mechanisms.

Active Mail allows users to customize (via the user interface) the set of events of which they want to be notified (e.g., text being added to a conversation or a document

being updated). When an event of interest occurs, the user hears a beep. If the client's window is iconified, the icon changes appearance. The Active Mail server keeps the states of all users, even if they are not logged in.

2.7 Network Notification Services

2.7.1 Simple Network Management Protocol (SNMP) [19]

The development of SNMP parallels the evolution of the Transmission Control Protocol / Internet Protocol (TCP / IP) protocol suite. A desire to monitor the performance of protocol gateways linking individual networks to the internet resulted in the development of the Simple Gateway Monitoring Protocol (SGMP), which can be viewed as the predecessor of SNMP. The need for changes and improvements of SGMP resulted in the Internet Activities Board (IAB), which was renamed in 1992 as Internet Architecture Board, recommending the development of an expanded Internet network management standard in a Request For Comment (RFC).

An SNMP-based network management system consists of three components – a manager, agent and a database – referred to as a Management Information Base (MIB). Although SNMP is a protocol that governs the transfer of information between its three entities, it also defines a client/server relationship. Here the client program is the manager, while the agent that executes on a remote device can be considered to represent a server; then the database controlled by the SNMP agent represents the SNMP MIB.

Manager The manager is a program that operates on one or more host computers. Depending upon its configuration, each manager can be used to manage a different subnet, or multiple managers can be used to manage the same subnet or a common network. The actual interaction between an end user and the manager is obtained through

the use of one or more application programs that, together with the manager, turn the hardware platform into a Network Management Station (NMS).

Through the manager, requests are transmitted to one or more managed devices. Originally SNMP was developed to used on TCP/IP networks and those networks continue to provide the transport for the large majority of SNMP-based network management products. SNMP can also be transported through NetWare IPX and other transport mechanisms.

Agents Each managed device includes software or firmware in the form of code that interprets SNMP requests and responds to those requests. The software or firmware is referred to as an agent. Non-SNMP-compatible devices can also be managed using proxy agent. The proxy agent can be viewed as a protocol converter since it translates SNMP requests into the proprietary management protocol of the non-SNMP device.

Although SNMP is primarily a poll-response protocol with requests generated by the manager, resulting in agent responses, the agent also has the ability to initiate an *unsolicited response,* which is an alarm condition resulting from the agent monitoring a predefined activity and noting that a predefined threshold was reached. Under SNMP that alarm transmission is referred to as a *trap*.

Management Information Base Each managed device can have a variety of configuration, status and statistical information that defines its functionality and operational capability. This information can include hardware switch settings, variable values stored as data in-memory tables, records or fields in records stored in files and similar variables or data elements. Collectively those data elements are referred to as the Management Information Base (MIB) of the managed device. Individually, each variable

data element is referred to as a *managed object* and consists of a name, additional attributes and a set of operations that can be performed on the object. Thus, the MIB defines the type of information that can be retrieved from a managed device and the device settings a user can control from a management system.

2.7.2 Remote Monitoring (RMON)

One of the problems associated with SNMP is the fact that its request-response (poll-select) operation, while having a relatively minor effect on the utilization of the bandwidth of a LAN, can result in the significant degradation of lower operating rate WAN bandwidth when monitoring geographically separated networks. To solve this problem the Internet Engineering Task Force (IETF) developed the Remote Monitoring (RMON) network management standard.

RMON represents an extension of the network manager's operation to distant networks. At those networks, intelligent devices known as *probes* or RMON *agents* monitor the data flowing on the remote network, organizing it into information the manager can easily access and interpret, with SNMP used as the transport mechanism between the manager and agent.

RMON has event, alarm, filter and capture groups. The filter and capture groups provide the mechanisms to capture traffic from a network and then later retrieve the captured packets. The alarm group provides the mechanisms so that the value of any integer-type object on a probe may be monitored to determine if it has risen or fell bellow a threshold value. Exceeding thresholds on objects that monitor network statistics can indicate network faults. The event group provides mechanisms that are used to perform an action when an event occurs. Examples of events are a threshold being exceeded or a packet match occurring. The actions include logging the event or sending an event

notification. The event and alarm groups provide general capabilities that are useful in devices other than probes [20].

2.7.3 Internet Control Message Protocol (ICMP)

Internet Control Message Protocol (ICMP) provides a modest number of basic control messages for error reporting. Even though Internet Protocol (IP) and ICMP are both part of the internet layer, ICMP uses the delivery services of IP.

The Internet Control Message Protocol (ICMP) group is responsible for handling error and control messages normally generated by gateways and hosts to reports problems to the originators of data-grams. It provides statistics and error counts for the ICMP protocol [21].

2.8 Unix Notification Mechanisms

This section briefly describes some of the notification mechanisms developed for the Unix environment.

2.8.1 biff

The biff gives notice of incoming mail messages. It turns mail notification on or off for the terminal session. With no arguments, biff displays the current notification status for the terminal. If notification is allowed, the terminal rings the bell and displays the header and the first few lines of each arriving mail message. Biff operates asynchronously. A "biff y" command can be included in user's ~/.login or ~/.profile file for execution when user logs in.

2.8.2 ps

The ps command prints information about active processes. Without options, ps prints information about processes associated with the controlling terminal. The output

contains only the process ID, terminal identifier, cumulative execution time and the command name. Otherwise, the information that is displayed is controlled by the options.

2.8.3 who

The who utility can list the user's name, terminal line, login time, elapsed time since activity occurred on the line and the process-ID of the command interpreter (shell) for each current UNIX system user.

2.8.4 cron

The cron command starts a process that executes commands at specified dates and times. It captures the output of the job's stdout and stderr streams and if it is non-empty, mails the output to the user. If the job does not produce output, no mail is sent to the user.

2.8.5 Zephyr

Zephyr is a notice transport and delivery system developed at MIT, which runs under 4.3BSD Unix. A notice transport and delivery system is a method of getting small quantities of time sensitive information efficiently from one client (or server) on a network to another. The object is to accomplish this with the highest possible fan-out (i.e., client to server ratio) while maintaining both network and server performance.

Zephyr is a multi-cast notice transport and delivery system based upon an authenticated datagram protocol. Localized Zephyr servers provide routing, queuing and dispatching services to clients, which communicate with them via the Zephyr Client Library. Two special purpose Zephyr clients, the WindowGram client and the HostManager client provide user and client host communication support.

Zephyrd servers run on designated server machines. These servers maintain a database of subscriptions and locations for every user using Zephyr. The servers stay in

contact with one another and provide a reliable backup system (via duplication) in the event of network failures.

Each client machine on the network runs a zhm HostManager client program that is the link between the Zephyr servers and the users. User programs send notices to the HostManager, and the HostManager forwards these notices to the nearest server for action. The HostManager is responsible for ensuring that the notices reach a server and for finding a new one if its server fails to respond.

Each user on the network usually runs a WindowGram client program automatically upon login. Zwgc displays notices to the user and handles user responses. Only notices to which the user has subscribed will be sent to the WindowGram client.

Subscriptions are handled through the zctl program. This program allows the user to add or delete subscriptions from Zephyr, to add the subscriptions to a file and to perform other miscellaneous functions.

The zephyr system consists of following suite of programs:

zaway	tell other people via Zephyr that the user is not around
zctl	zephyr control program
zleave	notify the user via Zephyr when the user have to leave
zlocate	find a user using Zephyr
znol	notify via Zephyr upon login or logout of interesting people
zwgc	Zephyr Windowgram Client program
zwrite	write to another user via Zephyr
zmailnotify	retrieve mail headers from post office and transmit via Zephyr
zhm	Zephyr HostManager
zephyrd	Zephyr server daemon
zstat	display Zephyr statistics
zpopnotify	notify users of newly spooled mail via Zephyr

Zephyr's delivery services are discussed in the next section.

2.8.6 syslogd

Syslogd reads and logs messages into a set of files described by the configuration file /etc/syslog.conf. Each message is one line. A message can contain a priority code, marked by a number in angle braces at the beginning of the line. Priorities are defined in <sys/syslog.h>. Syslogd reads from the UNIX domain socket /dev/log, from an Internet domain socket specified in /etc/services, and from the special device /dev/klog (to read kernel messages).

Syslogd configures when it starts up and whenever it receives a hang-up signal. Lines in the configuration file have a selector to determine the message priorities to which the line applies and an action. The action fields are separated from the selector by one or more tabs.

Selectors are semicolon separated lists of priority specifiers. Each priority has a facility describing the part of the system that generated the message, a dot and a level indicating the severity of the message. Symbolic names may be used. An asterisk selects all facilities. All messages of the specified level or higher (greater severity) are selected. More than one facility may be selected using commas to separate them.

For example, *.emerg;mail,daemon.crit, selects all facilities at the emerg level and the mail and daemon facilities at the crit level.

The additional facility “mark” has a message at priority LOG_INFO sent to it every 20 minutes (this may be changed with the –m flag). A facility field containing an asterisk does not enable the “mark” facility. The level “none” may be used to disable a particular facility.

For example, *.debug;mail.none sends all messages except mail messages to the selected file.

2.9 Unix Delivery Mechanisms

There are different mechanisms available that allow users to send text messages to one another, for example electronic mail and zwrite. These delivery mechanisms (email and zwrite) can be used not only by users to communicate to one another, but also by applications to communicate with users. Electronic mail has been used as a delivery mechanism to send notifications to users by different groupware applications, like Quilt, DCS v.1 and InConcert.

Delivery mechanisms can be classified based on the way they present the information to users as intrusive or non-intrusive mechanisms. Intrusive mechanisms are those that, without user intervention, show the message to the user (e.g., zwrite); non-intrusive mechanisms are those that require the user to access the message.

2.9.1 Electronic Mail

Electronic Mail is one of the better-known and broadly used mechanisms to send and receive text messages. Mail handlers provide a variety of flexible, comfortable and interactive user interface that allow users to compose, send and receive messages; they typically provide facilities to browse, display, save, delete and respond to messages, as well as to store the received messages. Mail is a non-intrusive delivery mechanism; messages are stored in the user's inbox and users can read them at any time.

Mail handlers use *sendmail* as the transport service to deliver the messages to the final users. Sendmail is used to send preformatted messages over the internet. With no flags, sendmail reads its standard input up to an EOF (End Of File) or a line with a single dot and sends a copy of the letter found there to all of the addresses listed. It determines the network to use based on the syntax and contents of the addresses.

Simple Mail Transfer Protocol (SNTP) is a TCP/IP protocol used in sending and receiving e-mail. However, since it is limited in its ability to queue messages at the receiving end, it is usually used with one of two other protocols, POP3 or Internet Message Access Protocol (IMAP), which let the user save messages in a server mailbox and download them periodically from the server. In other words, users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving messages that have been received for them at their local server. Most mail programs such as Eudora let user specify both an SMTP server and a POP server.

X.400 is the messaging (notably e-mail) standard specified by the International Telecommunications Union-Telecommunication Standard Sector (ITU-TS). It is an alternative to the more prevalent e-mail protocol, Simple Mail Transfer Protocol. Because X.400 stipulates a number of possible address characteristics that SMTP does not, an X.400 address can be long and cumbersome. X.400 offers more capabilities than SMTP does.

2.9.2 Zwrite

Zwrite is a delivery mechanism built using the services provided by Zephyr Notification Services. It is used to exchange text messages. It assures that the message is sent, not that it will actually arrive or how many copies will arrive.

Zwrite is an intrusive delivery mechanism; it displays the message in the user's window or terminal. Users cannot save the received messages, once the message is read it is discarded. Moreover it requires the user to be logged in the system and running the zephyr client program zwgc.

Zephyr provides two methods to display the information: Xwindow system display or generic terminal. The user can select what method he prefers or he is able to

use; by default, if the user is running Xwindow, the zephyr system will use Xwindow, otherwise it will use the generic terminal.

2.10 Active Databases

Traditional Database Management Systems (DBMS) are passive. They only respond to external requests in the form of database operations, for example modify and delete. In contrast, active databases are capable of immediate response to different events. Chakravarthy classifies the events that can be monitored by an active database as database events (e.g., insert, modify), temporal events (i.e., absolute and relative events) and external events (events detected outside the scope of the DBMS, but that are processed by the DBMS). He states that "an active database management system monitors *conditions* triggered by *events* representing database events, temporal events or external events and if the conditions evaluate to true then the *action* is executed" [13].

2.10.1 High Performance Active Database System - HiPAC

HiPAC is an active object oriented database management system. It is implemented using ECA (Event Condition Action) rules. The developers of HiPAC claim that the ECA rules subsume most of the active DBMS function that were previously implemented by means of special purpose mechanisms.

The semantics for the ECA rules are "when the event occurs (is *signaled*), evaluate the condition; and if the condition is satisfied, execute the action (216) [22]." A rule is specified by different attributes: *event* (the event that triggers the rule, that can be database operations, temporal events, external notification or any combination of them); *condition*, that is a collection of queries that are evaluated when the rule is triggered by its event; *action*: the action that is executed when the condition is satisfied (the actions

can be database operations or external requests to application programs); *E-C coupling*, which specifies when the condition is evaluated relative to the transaction in which the event is signaled (it can be immediate, separate or deferred); and *C-A coupling*, which specifies when the action is executed relative to the transaction in which the condition is evaluated. The rules in HiPAC are defined as database objects and they are subject to operations that are common to database objects: creation, modification and deletion.

HiPAC manages four interface modules to interact with the applications. Two of those modules provide common database interaction: database operations to define the classes and operations on instances of the classes and transaction operations to create, commit and abort transactions. The other two interface modules are the event operation, which allows applications to define an signal their own events and the application operation, which allows the DBMS to make requests to the application program as part of the actions executed when a condition for an event is satisfied.

HiPAC is composed of different functional components: The Object Manager, which provides object-oriented management; the Transaction Manager, which provides nested transactions; the Event Detector, which detects and signals primitive events to the Rule Manager; the Rule Manager, which maps events to rules; and the Condition Evaluator, which evaluates rule conditions.

HiPAC specifies a new paradigm of interaction between the application and the DBMS: control logic is encoded in the rules rather than software. This approach tends to simplify the software but it also produces large sets of rules, which may lead to unexpected interaction among rules.

2.10.2 Ariel

Ariel is “complete implementation of a relational DBMS with a rule system that is tightly coupled with the query processor” (22) [23].

The most important feature, also the research focus of Ariel developers, is the rule system it provides. The Ariel rule language (ARL) is a production-rule language improved to allow the definition of rules with conditions based on patterns, events and transitions. Ariel rules allow users to maintain database integrity constraints, monitor database states and build knowledge-based system applications.

The form of an Ariel rule is as follows:

```
define rule rule-name
[priority priority-val]
[on event]
[if conditon]
then action
```

The priority of the rule is used by the system to determine the order for the execution of the rules when more than one rule has to be run. The event determines under what circumstances the rule has to be triggered. Ariel manages different kinds of events related to database operations, time events and database transition conditions. Different actions can be defined; they can be a single database operation (data manipulation or definition statement) or they can be a compound command including a block of single database operations. Moreover, Ariel supports an external function interface that allows the user to call an external function as part of the action of the rule.

The combinations of the *on* and *if* clauses allow the user to define a variety of situations for triggering rules. *On* conditions respond to events; for time events the database generates a time event every second and the rules with the current time are triggered. An *if* clause is used to test the condition of the database when an update

transaction occurs; the rule is triggered if the condition is satisfied. Combinations of *on* and *if* clauses allow users to test the condition of the database when a given event occurs. Transition conditions use a special key word: *previous*; it allows comparison of two consecutive database states.

The rules in Ariel are processed using a strategy called the *recognize-act cycle* that consists of three steps: *match*, *conflict resolution* and *act*. The match step determines what rules must be run, the conflict resolution step determines the order of rule execution and the act step executes the statement indicated by the rule. This cycle is repeated as long as there are rules to execute.

2.11 Motivation For Notification Services

This section summarizes and discusses the motivation for Notification Services.

As seen above, group work requires a group's members to be aware of the actions taken by other members since those actions may affect their own work. Durish showed that information sharing, knowledge of group and individual actions and coordination are central to successful collaboration. Although Durish evaluated collaborative text editing, he affirmed that the "awareness information is always required to coordinate group activities, whatever the task domain" (107) [5].

Notification mechanisms have been used to provide information about actions taken by members of the group, as well as to inform them of changes in the shared environment. For example, notification is used to alert other users of modifications made by a user to a document or that a new task has been added to the user's list of pending activities. However, most of the groupware applications use notification mechanisms to support synchronous interaction among users, but they have only rarely been used to

support asynchronous interactions. Moreover, the notification services of the evaluated groupware application give little flexibility to the user to determine the set of events that may interest him and how he wants to be notified of those events. Such flexibility can aid in supporting the individual's needs for information about the collaborative process, which varied from person to person. Flexible notification services should also have mechanisms to assure that any information access constraint is observed.

Conferencing systems as Mermaid and DCS v.1 allow users to know what other members are doing while they are participating in the conference, but once the user leaves the active session no support is provided to inform him about the actions going on in the conference. Similarly, collaborative text editor as Mace and Grove support only synchronous collaboration and do not offer support to group's members that are not participating in the active session.

Other systems as Conversation Builder offer notification facilities but they are fixed, not allowing the user to modify them. On the other hand, there are systems that allow users to set the events that interest them: Quilt, InConcert and Active Mail.

Quilt supports asynchronous collaboration and uses notification and triggers to support the collaborative process. Quilt users can set triggers so that they receive notification when the document is changed. However, users are notified of the annotations made in the document if the person making the annotation sends them via email, otherwise the user must access the document to discover the annotations made to it.

Inconcert's trigger model provides a general and powerful mechanism for responding to events, but it requires some sophistication and programming on the part of the user.

Active Mail allows users to customize (via the user interface) the events upon which they want to be notified. When such an event occurs the users hears a beep. However if the user is not logged into system, the user will not be aware of the changes until he logs into the system again. However, the system keeps state information regarding al users, even if they are not logged in. Active Mail does not offer the alternative to the user to select the method of notification; for example, select to be notified by email.

Previous discussions highlighted the general need for a more flexible notification mechanism that allows users to be aware of the actions taking place in the active session even if they are not participating on it. This work aims to develop a system to provide notification services to the University of Florida Distributed Conferencing System.

2.12 Summary

The objective of this chapter was to provide a quick tour through CSCW and evaluate the notification mechanisms used by some groupware applications, as well as to provide a basic understanding of facilities and services that were considered during the design phase. A brief explanation of CSCW and Groupware was provided, highlighting the role that notification plays in the coordination of group work. An evaluation of different types of groupware applications was made: conferencing systems (DCS v.1 and Mermaid), editors (Mace and Quilt), workflow (InConcert) and general systems (Conversation Builder and Active Mail); the notification mechanisms used are intended

for supporting synchronous collaboration and those mechanisms provided for supporting asynchronous collaboration were limited, e.g., they covered few events, provided no flexibility to the users or required some sophistication and programming on the part of the user.

The next chapter presents the complete formal requirements for Notification Services on DCS v.2 system.

CHAPTER 3
SYSTEM REQUIREMENTS

The main function of the Distributed Conferencing System (DCS) is to support distributed conferences, distributed applications and diverse distributed activities over a Wide Area Network (WAN). DCS v.2 was conceived to provide a broader functionality and support that was not provided by DCS v.1. DCS v.2 can be categorized as a groupware that assists cooperative work among remote members of a conference by providing services that enable them to communicate and make co-decisions, among other things.

Multiple services will conform the entire system. Notification services constitute one of those DCS services. DCS v.2 daemon potentially provides all the following DCS services. The control service is a mandatory one and the others are optional.

1. **Control Service**. This service provides control primitives that allow the server to change its set-up or effect changes in other servers.
2. **Database Service.** This service provides the main application programming interface for all the file handling within DCS (create, open, close, retrieve, insert, update, delete and destroy).
3. **Access Control Service.** Conference access policies are managed through this service. It provides mechanisms for registering in a DCS conference the roles, members, objects, applications, permissions and actions to objects and classes.
4. **Secure Communication Service.** This service allows for multi-level secure communication and authentication between DCS processes.

4. **Secure Communication Service.** This service allows for multi-level secure communication and authentication between DCS processes.
5. **Conference Service.** Through this service, users can create, open, close, join, leave and destroy conferences. A conference is the main organizational structure in DCS v.2 and is defined as a set of members bound to roles, a set of shared objects and a set applications with the respective policies for controlling their access. Roles determine privileges of members to handle objects and applications. The access policies are described by relationships between members, roles, object and applications.
6. **Notification Service.** Through this service, users can request notification for certain events as well as specifying the way and frequency they prefer notification (through the Request Manager). A message deliverer can be used to send messages to particular conference members or to members bound to a role.
7. **Decision Support Service.** This service allows group decisions to be made within DCS. It supports online voting and flexible automated reporting, as well as multiple decision processes at the same time.

We can see that DCS v.2 facilitates cooperative work by providing services that support communication (notification service), collaboration and co-decision (notification and decision support services) among distributed users.

This chapter describes some related terms and gives the overview of the functional and non-functional requirements.

3.1 Notification Definitions Relative to DCS V.2

The following is the basic terminology related to Notification in DCS V.2.

1. **DCS Domain** constitutes a single instantiation of DCS that may cover different administrative domains and manage multiple conferences.
2. **Administrative Domain** is a set of resources under a single administration.
3. **Conference** is a tuple consisting of members, objects, applications, roles and allowable member-role bindings.
4. **Objects** are the class of passive entities that can be acted upon by subjects. Objects may be non-atomic.
5. **Applications** are the class of entities that a subject must use to act upon an object. Subjects use applications to act upon objects.
6. **Roles** are labeled set of privileges which may be bound to a subject. A subject has no privileges unless bound to a particular role.
7. **Member** of a conference is a DCS user that is allowed to bound his subject to a role within that conference.
8. **Active Member** is a member who has an instantiated subject, which is bound to a role in a conference.
9. **Inactive Member** is a member of a conference who does not currently have an instantiated subject bound to a role.
10. **User's Home Domain** is the administrative domain where the user has defined his account.
11. **Action** is anything that accesses objects.
12. **Event** is a state change.

3.2 Notification Specification

Notification Services (NS) in DCS v.2 is a tool for DCS's users that allows them to be aware of actions that occur in a conference without necessity of being active in that conference.

The system must be flexible, allowing DCS members to select the event or events of their interest and how they want to be notified. The user will be able to specify what method of notification requires, the information required about the event, and any time constraints that apply.

This section explains the type of events NS will manage, the notification methods and message formats that will be supported and the facilities to specify time constraints.

3.2.1 Events

NS will manage the following events.

1. A user becomes member of the conference.
2. A user loses membership to the conference.
3. A user loses access to a role.
4. A new role is added to the user's role list.
5. A member joins the conference.
6. A member leaves the conference.
7. Any user accesses a conference's object.
8. A specific user accesses a conference's object.
9. Any user modifies a conference's object.
10. A specific user modifies a conference's object.
11. A decision is reached.

12. A user needs to participate in a decision process; this event is raised when the user's vote is required to make a decision and the user is not actively participating in the conference.
13. An error occurs in the system.
14. A new conference is been created.
15. A new application is available.
16. An application is deleted from the list of available applications.

Only members of the conference can request notification regarding events going on in the conference and the information about the events the user may request is determined by the access rights of the role bound to the user's subject. There are some events (request for vote, loss of role, new role and loss of membership) of which the affected users must be informed by default.

3.2.2 Method of Notification

NS will offer different notification methods, allowing the user to select the method that better fits his requirements.

1. Instant Message: NS displays the notification message in the user's conference window. In this case the user must be active in the conference to be notified, otherwise the user will not be notified.
2. Asynchronous, on-line notification service outside DCS: in this case the user does not need to be active in the conference. However, the user must be reachable using a synchronous communication mechanism outside DCS, for example z-write. This requires the user to be logged into the computer and registered to receive z-write messages.

3. Asynchronous, off-line notification service outside DCS: in this case the user does not need to be active in the conference nor logged in the computer for the system to deliver the notification message. The system will deliver the message using an asynchronous notification mechanism outside DCS, for example, e-mail.
4. Selective: in this case the system will try to deliver the message by the fastest possible method to reach the user. NS will first try the Instant message, if that does not work, it will try to use the asynchronous, on-line method outside DCS and if that does not work either, the system delivers the message via the asynchronous, off-line mechanism.
5. Log file: in this case the notification messages are stored in a file. This option allows a user to keep a log of the most relevant events that happened in the conference. For example, it will allow us to make evaluations of the usage of the system that will be useful for future extensions to the DCS system. The user when making the request indicates the file where the messages are stored. The file must be located in the conference directory in the user's home domain.
6. Port: in some cases the message needs to be sent to a particular port. The user selects the choice of sending a message to a port and specifies the port number.

3.2.3 Time Constraints

The system will provide support to select the times of interest for receiving the notification messages. This facility of the system will allow users to set the time interval, i.e. the period of time during which they wish to be notified. The user can also specify

time granularity, which is the frequency with which the user wishes to receive notification messages.

Time interval of interest

The user should be able to specify the period of time over which he is interested in the information. This option allows the user to identify events that are relevant during a period of time, and only during that period the system will notify the user of them. For example, a user might want to know if someone accesses a document during last week.

It will be possible to set different time intervals:

1. hours,
2. days,
3. weeks,
4. months,
5. until an specific time and
6. endless (as long as the conference exist, or the user looses his rights in the conference).

This time will be counted from the moment the user asks for the service.

A user's request is valid as long as it does not expire and there is no change of the user's access rights to information about the event. For example, if the user loses membership in the conference, his requests are no longer valid.

Time Granularity

A user will be able to specify with what frequency he wants to be notified about a specific event. Thus, the user can avoid receiving a large number of notification messages if an event occurs frequently in the conference. For example, if a user wants to be

informed when a document is accessed but the frequency of accessing this document is every minute, the user may not want to be notified when accesses occur but would prefer an event summary after an hour.

It should be possible to set any of the following time granularities:

1. immediately,
2. minutes,
3. hours,
4. days,
5. weeks,
6. months and
7. at the end of the interval of interest

The times for delivering the messages will be calculated based upon the time interval and the time granularity. The messages will be sent when a time equal to some multiple of the time granularity selected has elapsed since the time the user made the request. This divides the time interval of interest into sub-intervals with duration equal to the time granularity, and the messages are sent at the end of each sub-interval. When the last delivery time falls outside the time interval, the last message will be delivered at the time specified but it will only contain information about the events that occurred within the time interval selected by the user.

3.2.4 Format of the Message

This issue is related to what information the user wants to receive once the event has occurred. For example, a user could be interested in knowing only that someone accessed the document or he could be interested in knowing who accessed it and when. The user's access rights will determine what information the user is allowed to see.

The following formats should be offered:

1. Simple: stating only that one or more instances of the event took place.
2. Detailed: including details as:
    (a) what conference's objects were accessed or modified, when and by whom,
    (b) who joined or left the conference and when,
    (c) what decisions were taken and their results,
    (d) what is the faulty system and when it failed,
    (e) what access rights were denied to the user and when and
    (f) what applications are available or no longer available.

3.2.5 User Interface

The user interface will provide the users with the necessary information for them to make a request, modify or delete a previous request and see the previous requests they made.

The user interface will offer to the users:

1. the list of different events,
2. the list of members of the conference
3. the list of the conference's objects,
4. an option to set time interval,
5. an option to set time granularity,
6. the list of different message formats and
7. the list of different notification methods.

The user will be able to choose from each list and set the values required to ask for notification. Also, a list of previous requests will be displayed so users can see

previous requests and choose from the list those request that need to be modified or deleted.

The user interface should always present to the user only those events that the user is allowed to access.

3.3 Non-functional Specification

3.3.1 Security Constraints

Notification Services can only be accessed by DCS's users. A user can request notification services relative to a conference only if he is a member of that conference. The user must be active to make the request and the role must have the correct permissions for the request to be honored. The role bound to the user's subject will determine his privileges over conference's objects and the kind of information for which the user can ask. The user's privileges determine the list of events the user may access, as well as the type of information he may access about those events, for example, whether the user has access to detailed information about the event or not.

If a user's membership in a conference is terminated, then all notification requests made by the user are also terminated. Additionally, when a user is no longer allowed to bind his subject to a role and the user made notification requests using that role that is now denied him, the notification requests made when bound to that role will be terminated. The user will be notified that the role was deleted from his role's list and his entries in the notification database will be deleted.

3.3.2 Information Storage

All requests made by the users must be stored in permanent storage. All storage space required by NS must be part of the space resources assigned to the conference.

Similarly, the files used to store the notification messages must be located within the conference directory and must have the correct permissions for NS to access them to append the notification messages.

3.3.3 Interaction with Other DCS Services

Notification interacts with other DCS's services in two different ways:

1. Offering services. NS should provide a clean interface to other DCS services so that they can use NS to send information to users. Examples of how other services might use NS include the following:
    (a) Fault Tolerance Services can use NS to inform users about a fault that requires human intervention.
    (b) Decision Support System can use NS to inform users that are not active in the conference that their vote is required or that a decision has been reached.
2. Requesting Information. NS requires information about users access rights, users e-mail addresses, conference status and event occurrences. DCS's services upon which NS relies include the following:
    (a) Conference Control and Management informs NS about events that change the conference status: members joining and leaving the conference, modification to the user's role lists, DCS users becoming members of the conference and creation of new conference.

    The Conference Manage should also provide information about the current location of the active members for synchronous messages in the

conference, information about roles bound to the user's subject, domains where the conference is active, conference name or id, conference's object and communication services.

(b) Global Conference Directory Services (GCDS) provides to NS information about users e-mail, conference members and member's home domains.

(c) Fault Tolerance informs NS when a fault occurs in the system.

(d) Application Manager informs NS when a new application is available, when an object is accessed or modified and when an application is deleted.

(e) Decision Support System informs NS about the results of a motion and when to request a user to vote.

3.3.4 Hardware and Software Environment

The Notification Service is written using Java language and it runs on a Linux platform.

3.4 Summary

The purpose of this chapter was to present the formal requirements for Notification Services in DCS v.2. Definitions for the most frequently used terms have been provided. Following them, the functional and non-functional specifications have been presented.

CHAPTER 4
DESIGN

4.1 Introduction

Distributed Conferencing System (DCS) is composed of seven different modules as described in Chapter 3. Figure 4.1 shows the different components of DCS. Notification service is one of the modules of DCS. It is seen as a single functional module that provides services to other DCS component.

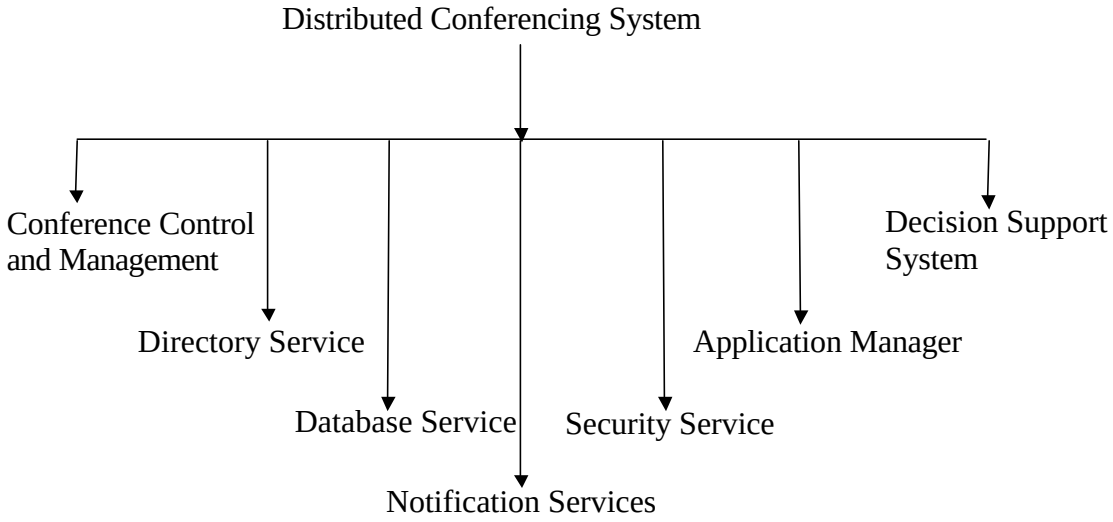

Figure 4.1 Components of DCS

4.2 Communication

Notification services communicate with users outside the DCS through two external delivery mechanism: zwrite and email.

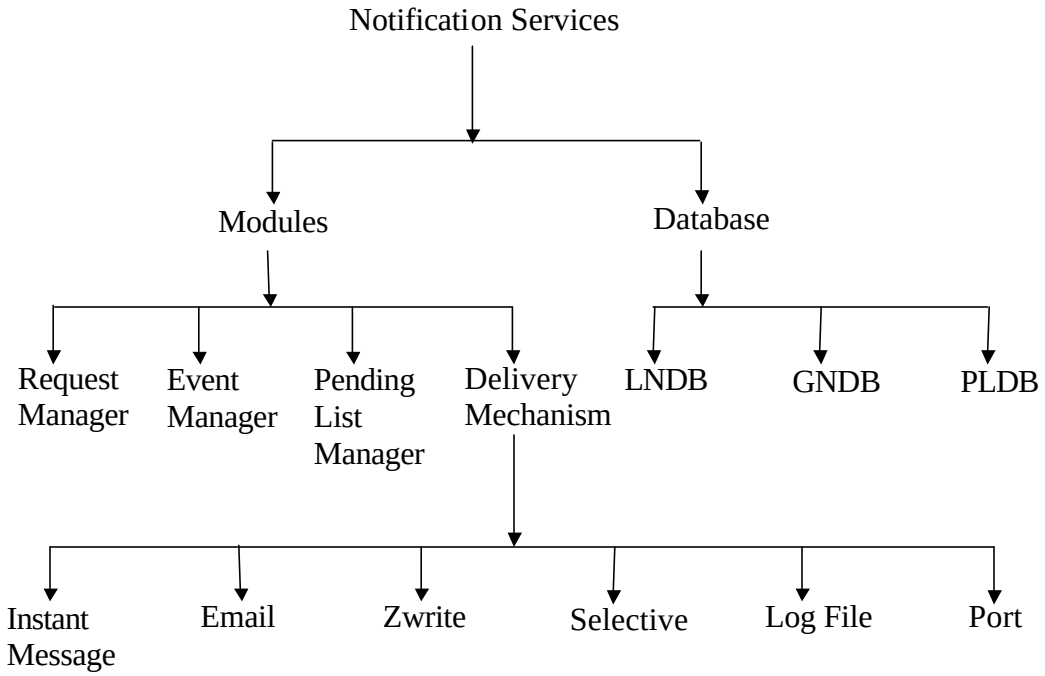

Figure 4.2 Components of Notification Services

1. Zwrite: When the user is not logged into DCS but he is logged into the system, zwrite delivers the message to the user from the sender via the zephyr notification service. Zwrite is used as the asynchronous on-line communication method to deliver the notification message to the end users.
2. email: When the user is neither logged into DCS nor into the system, email delivers the message to the user using the smtp port number 25. It is used as an asynchronous off-line communication method. It sends the message independent of the user's location.

4.3 Architecture

Notification service is divided into four modules and it uses three databases [Local notification database (LNDB), Global notification database (GNDB) and Pending list database (PLDB)]. Figure 4.2 shows the hierarchy of different notification service modules. Figure 4.3 shows different modules of notification service.

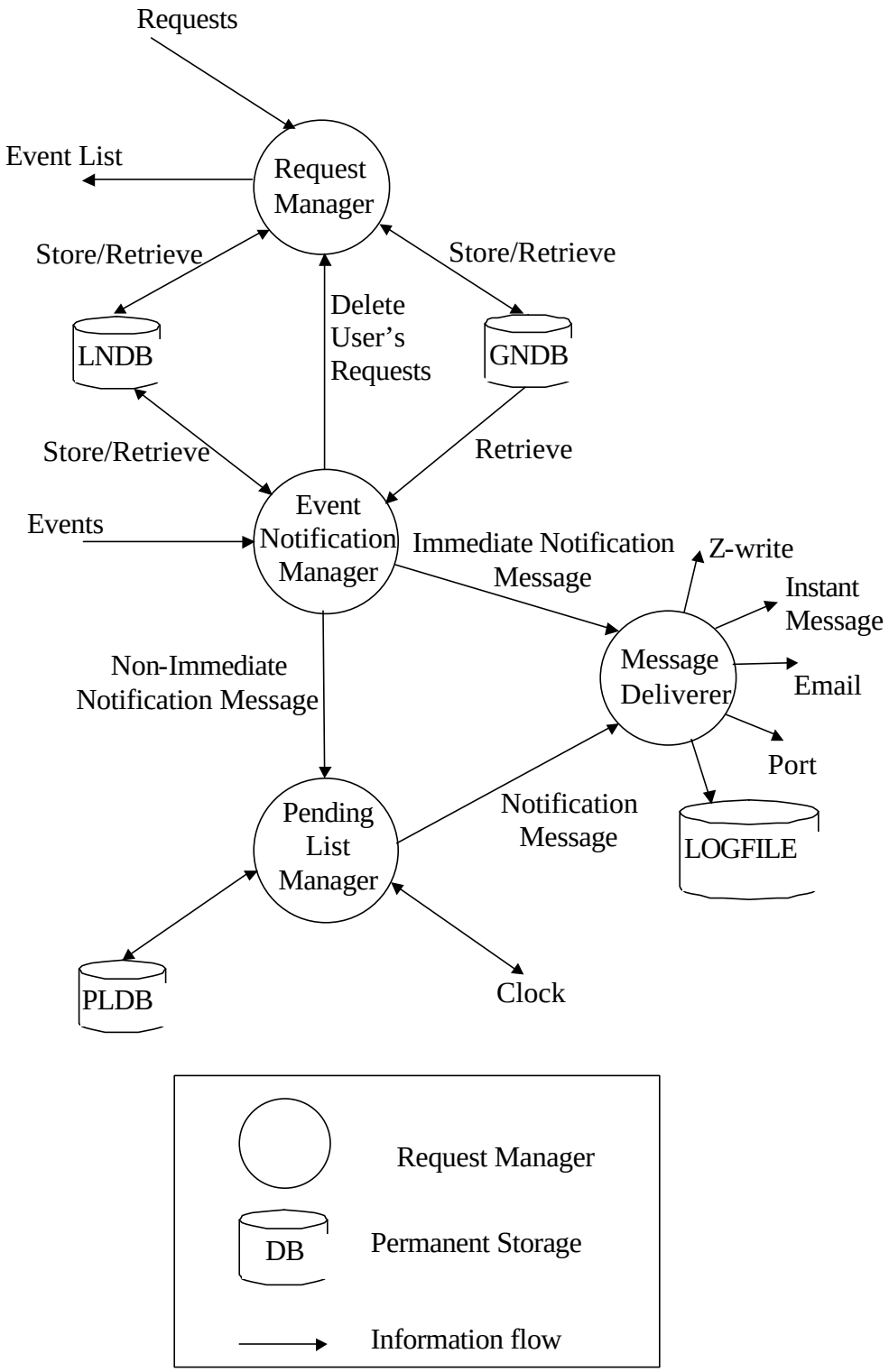

Figure 4.3 Modules of Notification Services

4.3.1 Local Notification Database

The Local Notification Database (LNDB) keeps detailed information about the requests made by *local* members of the conference. *Local* members of the conference are those DCS users who are members of the conference and their home domain corresponds to the domain where the LNDB resides.

There is an LNDB for each domain where the conference is instantiated. In each administrative domain there will be an LNDB for each conference instantiated in that domain.

For each LNDB, a set of counters (one for each eventId) is kept to save the number of requests placed per eventId. This information is used to update the GNDB, see section 4.3.2, Global Notification Database, and section 4.3.4, Request Manager.

Table 4.1. EVENT TYPES MANAGED BY NOTIFICATION SERVICES

EVENT_ID	EVENT
JOIN	A member joins the conference
LEAVE	A member leaves the conference
OBJECT_ACCESS	An object is accessed
OBJECT_MODIFY	An object is modified
OBJECT_DELETE	An object is deleted
DECISION_REQ	Request for Decision
DECISION_DONE	Decision done
NEW_CONF	New conference created
DELETE_CONF	Conference deleted
FAULT	Fault in the system
NEW_ROLE	User has a new role in his role list
LOSS_OF_ROLE	User lost the role and it is deleted from the list
MODIFY_ROLE	Role’s access rights modified
NEW_MEMBER	New member in the conference
LOSS_OF_MEMBERSHIP	Member lost access to the conference
NEW_APPLICATION	New application is available
DELETE_APPLICATION	Application is no longer available
NEW_ADDRESS	User’s email address changed

The LNDB stores the following information :

1. **DCSID** is an integer that uniquely identifies the user, that made the notification request, within the DCS domain.
2. **Role** is an integer that represents the role to which the user's subject was bound when requesting notification.
3. **EventId** is an integer that identifies the event of interest. It can only take one of the values listed in table 4.1.
4. **DCSID_2** is an integer that uniquely identifies a user within DCS. It must have a valid value when the event of interest is JOIN, LEAVE, OBJ_ACCESS, OBJ_MODIFY, OBJ_DELETE, NEW_ROLE, LOSS_OF_ROLL, MODIFY_ROLE, NEW_MEMB, LOSS_OF_MEMBERSHIP or NEW_ADDRESS.
5. **ObjectId** is an integer that has a valid value when the event of interest is OBJ_ACCESS, OBJ_MODIFY or OBJECT_DELETE.
6. **Role_2** is an integer that identifies one of the roles defined in the conference. It must have a valid value when the event of interest is NEW_ROLE, LOSS_OF_ROLE or MODIFY_ROLE.
7. **ApplId** is an integer that identifies the application's type (i.e., editors, graphics). It must have a valid value if the event of interest is NEW_APPL or DELETE_APPL.
8. **Fault_type** is an integer that identifies the fault's type (i.e., network failure, system failure). It must have a valid value if the event of interest is FAULT.

9. **Time Interval** is a long integer that saves the expiration time (in seconds) for the request. Its values can be ENDLESS or greater than zero.
10. **Time Granularity** is a long integer that saves the granularity (in seconds) for the request. Its values can be IMMEDIATE or greater than zero.
11. **Method** is an integer that saves the method selected by the user. It can take any one value of NTF_EMAIL, NTF_ZWRITE, NTF_SELECTIVE, NTF_FILELOG, NTF_WINDOW and NTF_PORT.
12. **Format** is an integer that saves the format selected by the user. Its only possible values are NTF_SIMPLE, NTF_PERSONAL or NTF_DETAILED. NS to notify the user about personal events, for example, when a user gets access to a new role, internally uses NTF_PERSONAL.
13. **Email Address** is a string that saves the user's email address.
14. **Filename** is a string that stores the name of the file to store the notification message if the method selected is NTF_FILELOG.

Information is stored in LNDB based on the eventId. LNDB manages the following queries.

1. Read_Request(pattern). In this case the LNDB retrieves all requests that match the given pattern.
2. Add_Request(information about the request). In this case the requested information is added to the LNDB, in the appropriate location to maintain the requests ordered by eventId.

3. Modify_Request(information about the new request). In this case, time interval, time granularity, method and format (i.e., the request parameters) can be modified.
4. Delete_Request(pattern). In this case all requests that match the given pattern are deleted.

4.3.2 Global Notification Database

The Global Notification Database (GNDB) keeps information about the events requested per administrative domain. It is a replicated database that resides in each administrative domain where the conference is instantiated.

The GNDB is updated when a request for an event (that occurs for the first time) is added to a LNDB in the conference or when a unique or last remaining request for an event is deleted from a LNDB in the conference. The updates are done on a per administrative domain basis, the number of requests per event is maintained by each LNDB. Thus, when the event counter becomes zero or it goes from zero to one (the request is the last or the first one for the event), the GNDB is updated.

When an event occurs, the NS running in the same domain notices where the even took place. NS notifies its local users and informs only interested administrative domains. Those domains where no request for the event exists do not need this information. So by maintaining GNDB, it reduces the number of messages among administrative domains generated by NS.

GNDB stores the following information:

1. **EventId:** this is an integer that identifies the event of interest and
2. **Lcmid:** this is an integer that identifies an administrative domain.

Following two queries are performed in GNDB.

1. Add_Entry(eventId, lcmid) and
2. Delete_entry(eventId, lcmid).

4.3.3 Pending List Database (PLDB)

One of the responsibilities of NS is to deliver the messages, which are non-immediate. The user is allowed to select the time period by what he wants to receive messages. For this purpose, the information has to be stored somewhere and has to be retrieved when it's time to send the information to the user.

The Pending List Database (PLDB) keeps the notification messages that are to be delivered to the users at a later time.

Similar to LNDB there will be PLDB for each administrative domain, which stores the following information per message:

1. **DCSID** is an integer that uniquely identifies the user within DCS and indicates in this context to whom the message to be delivered.
2. **Role** is an integer that represents the role to which the user's subject was bound when requesting notification.
3. **EventId** is an integer that identifies the event of interest.
4. **DCSID_2** is an integer that uniquely identifies a user within DCS. It must have a valid value when the event of interest is JOIN, LEAVE, OBJ_ACCESS, OBJ_MODIFY, OBJ_DELETE, NEW_ROLE, LOSS_OF_ROLE, MODIFY_ROLE, NEW_MEMB, LOSS_OF_MEMBERSHIP or NEW_ADDRESS.
5. **ObjectId** is an integer that must have a valid value when the event of interest is OBJ_ACCESS, OBJ_MODFY or OBJ_DELETE.

6. **Role_2** is an integer that identifies one of the roles defined in the conference. It must have a valid value when the event of interest is NEW_ROLE, LOSS_OF_MEMBERSHIP or MODIFY_ROLE.
7. **ApplId** is an integer that identifies the application's type. It must have a valid value if the event of interest is NEW_APPL or DELETE_APPL.
8. **Fault_type** is an integer that identifies the fault's type. It must have a valid value if the event of interest is FAULT.
9. **Method** is an integer that saves the method selected by the user. It can take any one value of NTF_EMAIL, NTF_ZWRITE, NTF_SELECTIVE, NTF_FILELOG, NTF_WINDOW or NTF_PORT.
10. **Email Address** is a string that saves user's email address.
11. **Filename** is a string that stores the name of the file to store the notification message if the method selected is NTF_FILELOG.
12. **Delivery Time** is a integer that indicates the time (in seconds) when the message has to be sent to the user.
13. **Event Time** is an integer that indicates the time (in seconds) when the event occurred.
14. **Message** is a string that contains the message that has to be sent to the user, which is non-null.

The records are based on delivery time. When more than one event has the same delivery time, the order is determined by the time when the event occurred. The following two queries are performed in the PLDB:

1. Add_Entry(information about the message) and

2. Delete_Entry(pattern).

4.3.4 Request Manager

This module is responsible for managing the requests made by the users and creating the appropriate entries in the LNDB and GNDB. Figure 4.4 shows details of the Request Manager.

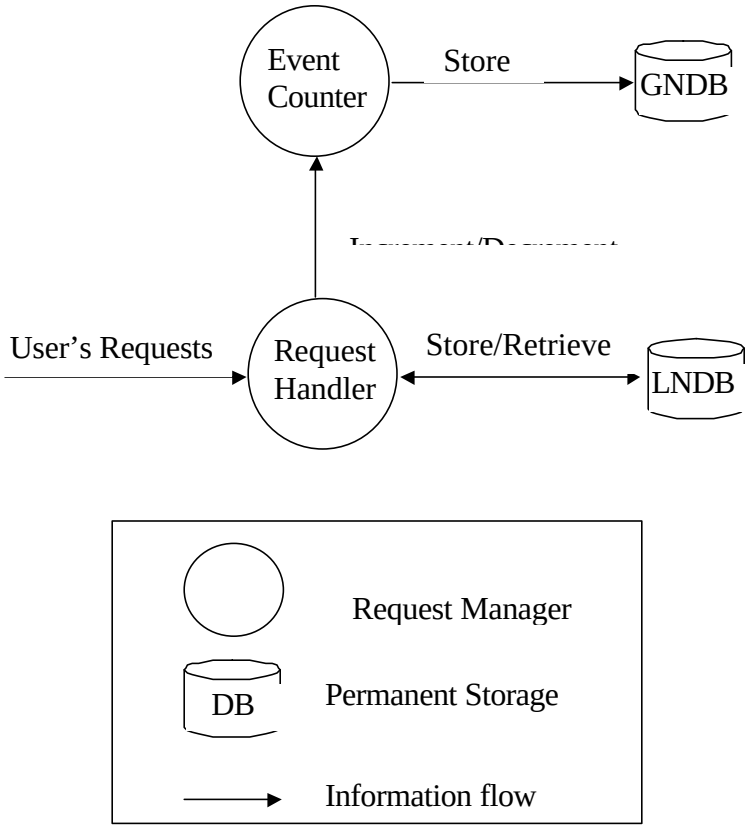

Figure 4.4 Request Manager

Event Counter keeps the track of number of requests per eventId in the LNDB. GNDB gets updated when the number of requests per eventId becomes zero or it changes from zero to one (the first or last remaining request).

Request Handler receives the requests, updates the LNDB and informs the *Event Counter* of those requests that result in an addition or deletion from the LNDB. Modifications to the requests need not be propagated to the event counter since they do not change the event requested, only some of its parameters.

Primitives

1. **addFunc()**

   addFunc() stores the user's request in the LNDB. addFunc() checks first if the request is valid. If the request has not been previously made, addFunc() determines the user's e-mail address, stores the user's request in the LNDB and informs the event counter. It will assure that the user does not make the same request twice using the same role. Moreover, it will also assure that requests that may produce two notification messages for the same event are not made. For example, if the user requests to be notified when *any* member of the conference leaves the conference, he cannot ask later to be notified if a *specific* members leaves the conference, or vice versa. These requests are consider equal, since they will generate two messages when the specific member leaves the conference. It is important to highlight that NS identifies a user as a dcsid-role pair since it clearly determines the set of privileges the user has. Thus, uniqueness of requests is assured based on the dcsid-role pair. One consequence this is that a user may make the same request twice if he binds his subjects to different roles. This case is left to the user since it simplifies the design without sacrificing functionality or fulfillment of the requirements.

2. modifyFunc()

modifyFunc() modifies the user's requests. This function checks the validity of the request. It modifies the entry in the LNDB if the request is valid, otherwise it returns with the appropriate error code. The user can only modify some of the attributes: time interval, time granularity, notification method and format. The rest of the attributes cannot be modified because they uniquely identify the request. If a user wishes to change any of those attributes he has to delete the original request and make a new one.

3. deleteFunc()

deleteFunc() deletes all the user's requests that match the pattern supplied. For example, if the user wants to delete all the requests for notification when a user leaves the conference, he only needs to indicate eventId = LEAVE and all those requests will be deleted, assuming that all the requests were made with the same role. This function deletes the request.

4.3.5 Event Notification Manager

The Event Notification Manager receives information about the events that occurred (locally or remotely) in the conference and creates the messages to be sent to the Message Deliverer or the Pending List Manager (immediate or non-immediate notification). Figure 4.5 shows details of the Event Notification Manager.

Message creator creates the final message to be sent to the user and sends it to the Message Deliverer or the Pending List Manager. The format of the messages created by the Event Manager for each eventId is described in Tables 4.2, 4.3 and 4.4.

Event Handler searches the GNDB and LNDB and propagates the information to the interested domains and the *Message Creator* to generate the messages to be delivering to each user interested in the event.

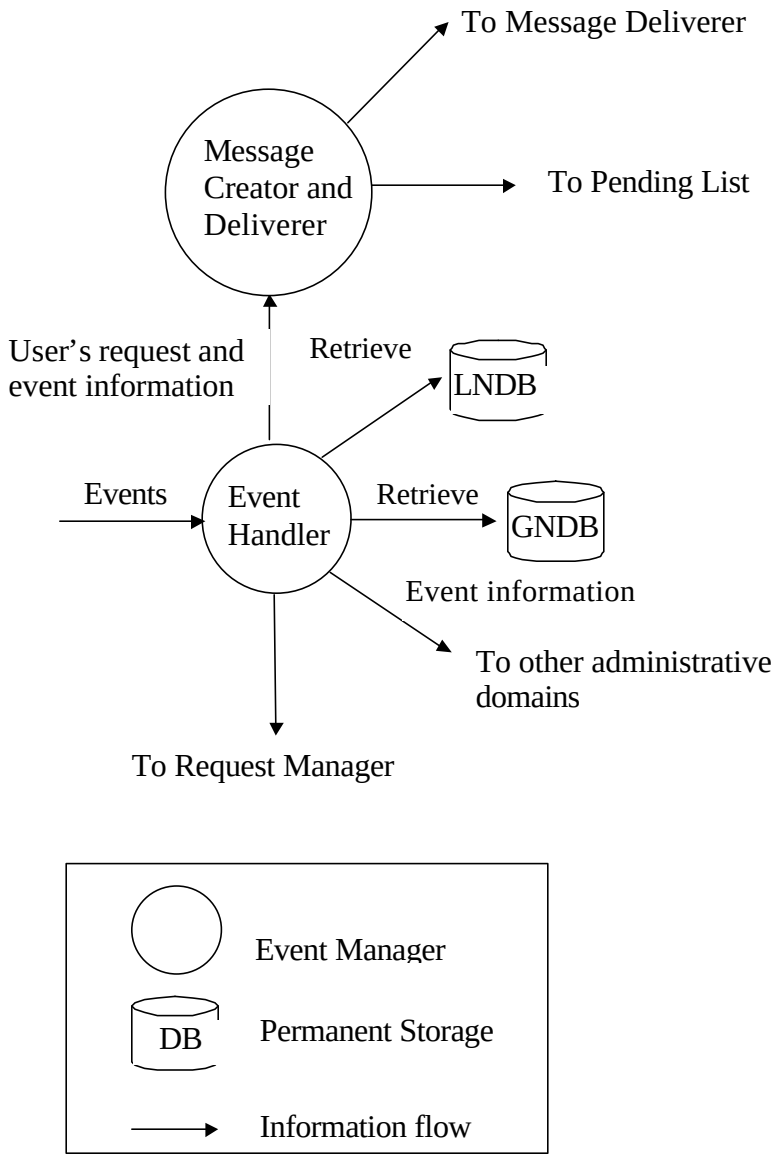

Figure 4.5 Event Manager

Table 4.2 SIMPLE MESSAGE FORMAT USED BY THE EVENT MANAGER

EVENT_ID	Simple Message Format
JOIN	Member _____ joined the conference
LEAVE	Member _____ left the conference
CREATE_ROLE	Role _____ is available
DELETE_ROLE	Role _____ is unavailble
NEW_ROLE	Member _____ gained access to role _____
LOSS_OF_ROLE	Member _____ lost access to role _____
MODIFY_ROLE	Privileges for role _____ changed
FAULT	Fault _____ occurred in the system
OBJECT_ACCESS	Object _____ was accessed
OBJECT_MODIY	Object _____ was modified
OBJECT_DELETE	Object _____ was deleted
DECISION_REQ	Your vote is required for motion _____
DECISION_DONE	Decision for motion _____ was _____
NEW_CONFERENCE	A new conference _____ has been created
DELETE_CONFERENCE	Conference _____ was deleted
NEW_MEMBER	Conference _____ has a new member
LOSS_OF_MEMBERSHIP	Member _____ lost access to conference
NEW_APPLICATION	New application _____ is available
DELETE_APPLICATION	Application _____ was deleted
NEW_ADDRESS	Member _____ changed the e-mail address

Table 4.3 DETAILED MESSAGE FORMAT USED BY THE EVENT MANAGER

EVENT_ID	Detailed Message Format
JOIN	Member _____ joined the conference with role _____ at time _____
LEAVE	Member _____ with role _____ left the conference at time _____
NEW_ROLE	Member _____ gained access to role _____ at time _____
LOSS_OF_ROLE	Member _____ lost access to role _____ at time _____
MODIFY_ROLE	Privileges for role _____ changed to _____ at time _____
FAULT	Fault _____ occurred in the system at time _____
OBJECT_ACCESS	Object _____ was accessed by member _____ with role _____ at time _____
OBJECT_MODIY	Object _____ was modified by member _____ with role _____ at time _____
OBJECT_DELETE	Object _____ was deleted by member _____ with role _____ at time _____
DECISION_REQ	Your vote is required for motion _____ by _____
DECISION_DONE	Decision for motion _____ was _____ and the votes were _____
NEW_CONFERENCE	A new conference _____ has been created since time _____
DELETE_CONFERENCE	Conference _____ was deleted at time _____
NEW_MEMBER	User _____ became member of the conference with roles _____ at time _____

Table 4.3 Contd. DETAILED MESSAGE FORMAT USED BY THE EVENT MANAGER

LOSS_OF_MEMBERSHIP	Member _____ lost access to conference at time _____
NEW_APPLICATION	New application _____ is available since time _____
DELETE_APPLICATION	Application _____ was deleted at time _____
NEW_ADDRESS	Member _____ changed the e-mail address to _____

Table 4.4 PERSONAL MESSAGE FORMAT USED BY THE EVENT MANAGER

EVENT_ID	Personal Message Format
LOSS_OF_ROLE	Your access to role _____ was revoked at time _____
NEW_ROLE	You have access to role _____ since time _____
LOSS_OF_MEMBERSHIP	Your access to the conference was revoked at time _____
NEW_ADDRESS	Your e-mail address has been changed to _____

Primitives

1. ntfEvent()

This function is used by other DCS services to inform NS about the occurrence of the events in the conference. When an event occurs locally, ntfEvent takes the information about the event, provides a mechanism to confirm that the raised event is valid and searches the GNDB to find the domains interested in the event. If there are requests in the local domain, ntfEvent also searches the LNDB for

users interested in it. If the event is lose role, new role, or lose membership, the affected user is notified immediately via e-mail (regardless of whether the user requested it) and all his requests that are not longer valid are deleted. A special case occurs when the e-mail address of a user is changed. In this case the event handler updates the information in the LNDB.

ntfEvent() checks the delivery time (immediate or non-immediate) of each of the requests made by users interested in the event and calls the appropriate function ntfImmediate() or ntfList(). If the time interval of the user's request has expired ntfEvent() deletes the user's request from the database.

4.3.6 Message Deliverer

The Message Deliverer sends the messages to the final users using the method selected by the user. See Figure 4.6.

Primitives

1. **ntfNotify()**

   This function receives the message to be delivered to the user and the method of notification and calls the appropriate function to deliver the message to the user (ntfWindow(), ntfZwrite(), ntfMail(), ntfSelective(), and ntfStoreF()).

2. **ntfWindow()**

   This primitive finds out whether the user is active or not in the conference. If the user is not active in the conference the message is discarded. If the user is active in the local domain the notification message is displayed in the user's announcement window.

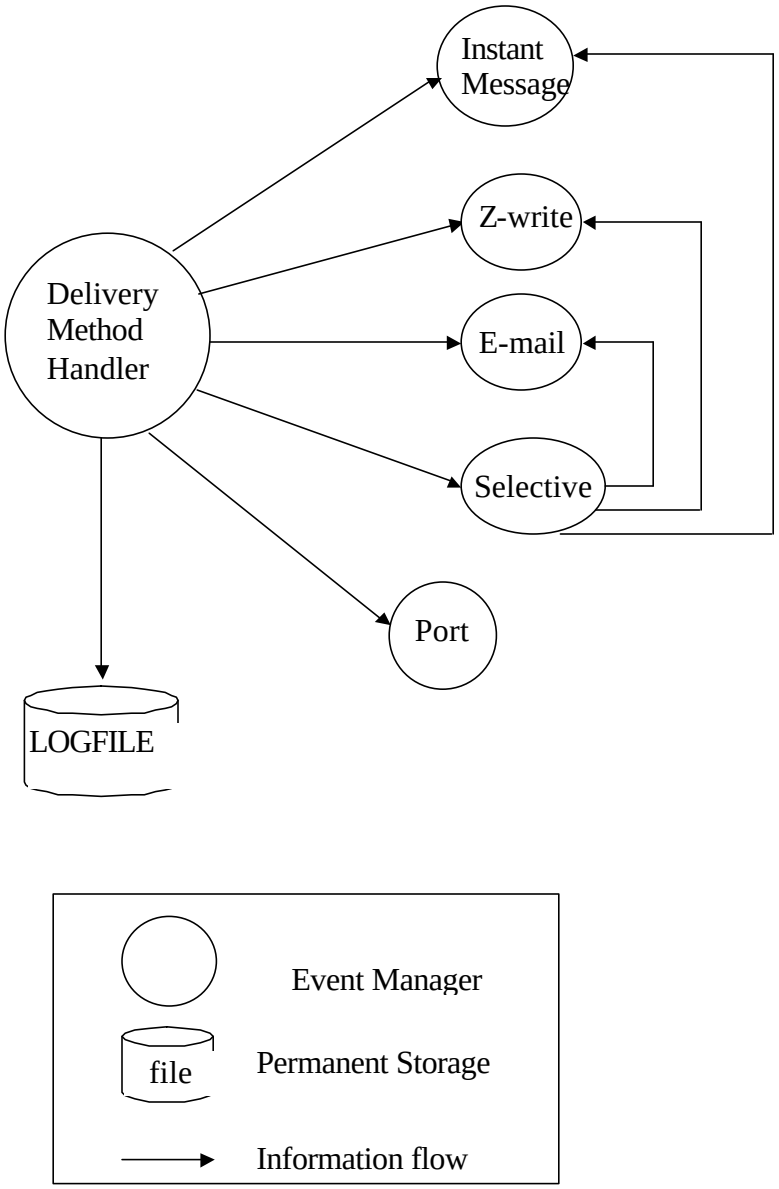

Figure 4.6 Message Deliverer

3. ntfZwrite()

This primitive determines where the user was last authenticated before sending the message. It gets the information about active members from the Conference Management. If the user was authenticated locally, the message is sent using zwrite. If the user was not authenticated locally, then it finds out where the user currently logged in and also checks if the remote site has a delivery module. If the

remote site has a delivery module than local delivery module talks with the remote one and also passes the other required parameters.

4. **ntfEmail()**

   This primitive sends the message using Simple Mail Transfer Protocol (SMTP) port number 25 to the email address stored in the request.

5. **ntfSelective()**

   This primitive will send the message by the fastest way to reach the user. The process followed by ntfSelective() is explained using the following algorithm.

   (a) if the user is not active in the conference goto (c);

   (b) if the user is active locally, display message in user's window;

   (c) find out where the user was authenticated most recently;

   (d) send zwrite to user and wait for return value;

   (e) if return value is OK, (end);

   (f) if return value of zwrite is not OK then send email.

6. **ntfStroreF()**

   In this case, the message is appended to the file indicated by the user when he made the request.

7. **ntfPort()**

   In this case, the message is sent to the port specified by the user when he made the request.

4.3.7 Pending List Manager

The Pending List Manager is responsible for saving the information about the notification messages in the PLDB. It checks the correctness of the information before it saves the new message information in the database, to assure that all the constraints are

observed. When the messages are to be sent to the final users, the Pending List Manager comply all messages that have to be sent to each user and delivers them to the Message Deliverer.

The Pending List Manager maintains a timer that indicates the delivery time for the message(s) that have the earliest delivery time. When a new message is saved in the PLDB, it checks whether the new message has an earlier delivery time or not. If the new message has the earliest delivery time, the timer is reset to the new delivery time.

When the timer expires each message whose delivery time is less than or equal to the current time is delivered to the user. The Pending List Manager complies in a single message all messages that correspond to the same event on a per user basis, and delivers them to the Message Deliverer. All entries that correspond to those delivered messages are deleted from the PLDB, and the timer is set to the new earliest delivery time.

4.4 Summary

This chapter described the detailed design for the Notification Services in DCS v.2 based on the requirements presented in Chapter 3. Different DCS components and components of Notification Services were discussed. The architecture and functional modules of the system that include Request Manager, Event Notification Manger, Pending List Manager and databases that include Local Notification Database, Global Notification Database and Pending List Database were explained in detail. The following chapter explains the implementation of the system.

CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Introduction

Distributed Conferencing System (DCS) v.2 uses a client-server architecture. It uses Remote Method Invocation (RMI) for that purpose. The DCS server is a regular LINUX program consisting of one or more DCS services. The following sections describe the important issues related to the implementation of Notification Services.

5.2 General

Notification Services are provided on a per conference basis. Each conference instantiated in DCS must have its set of NS databases (LNDB, GNDB and PLDB), event counter and timer. The NS databases are accessed based on their name, which is unique, and is formed by the generic name of the database (LNDB, GNDB and PLDB) and the confId (an integer that uniquely identifies a conference, maintained by Conference Control and Management). For example, for conference 1, the database names are LNDB.1, GNDB.1 and PLDB.1.

5.3 Message Size, Time Manipulation and Timers

The size of the messages created by the Event Manager is of variable size. Messages that are sent in a non-immediate fashion also have variable size, since they compile information about what has occurred over a given period of time. No maximum message size is set to these messages since it depends on the number of times an event occurs in the conference and the time granularity the user selects.

Different time values are managed by the Notification Services: the request time, time interval, time granularity and delivery time. The value of the request time and the time interval depends upon the domain they are in and the time zone that domain follows. Time granularity is represented in seconds. Therefore, times can be seen as values within a single time scale measured in seconds.

5.4 Communication Between Client-Server

Client-server communication is achieved using Remote Method Invocation (RMI). Here the client is a Graphical User Interface (GUI), which is developed using Java Swing. The user will have different choices about event of interest, time interval, format of the message, method of message delivery etc. All this information is passed to the server using the interface. The server will take appropriate actions after it receives the information from the client.

Remote Method Invocation [24]

Remote Method Invocation is a facility that allows Java programs to call certain methods on a remote server. The RMI interface lets Java objects on different hosts communicate with each other. A remote object lives on a server. Each remote object implements a remote interface that specifies which of its methods can be invoked by clients. Clients can invoke the methods of the remote object almost exactly as they invoke local methods.

From the programmer's perspective, remote objects and methods work just like local objects and methods. A *remote object* is an object whose methods may be called by a different Java virtual machine than the one where the object itself lives, generally one

running on a different computer. The fundamental difference between remote objects and local objects is that remote objects reside in a particular virtual machine.

5.5 Interface

Two kinds of user interfaces are provided: the Application Programmer's Interface that allows programmers to use notification primitives to develop various DCS services and DCS's Graphical User Interface that allows DCS users to make requests for notification.

5.5.1 Application Programmers' Interface

The Application Programmers' Interface (API) is a set of library routines in the form of an object link library. This object library allows the application programmers to use the services provided by Notification Service. Decision Support Systems and Fault Tolerance are examples of services that will use the services provided by NS.

The following discusses the calling interfaces for various API calls. The description of the functionality of this functionality of these primitives is provided in Chapter 4 and Appendix A.

int addFunc(String event1, String method1, String format1, String interval1, String granularity1, String fault1, String member1, String object1, String role1, String appid1, String filename1, int timeinterval1, int timegranularity1)

int modifyFunc(String event1, String method1, String format1, String interval1, String granularity1, String fault1, String member1, String object1, String role1, String appid1, String filename1, int timeinterval1, int timegranularity1)

int deleteFunc(String event1)

int ntfEvent(eventStruct ev, int location)

int ntfNotify(String msg, deliverStruct del)

The classes used by these functions are eventStruct and deliverStruct.

```
class eventStruct{
        int eventId;      /*to indicate the event                                  */
        String time1;     /* to indicate at what time ocurred the event            */
        int objectId;     /*to indicate object                                     */
        int dcsId;        /* dcsId of user that lost role, or has new email        */
        int role;         /* to indicate what role was lost                         */
        int fault;        /* to indicate faulty system, process?                    */
        int faultType;    /*to indicate type of failure                            */
        int applId;       /*to indicate application id                             */
        int confId;       /*to indicate conference id                              */
        String result;    /* to indicate result of voting, motion description.     */
}
```

The eventStruct class is used by DCS services and applications to inform NS of the events. Based on the eventId the information passed to Ns varies.

```
class deliverStruct{
        int dcsId;            /* to whom                                            */
        String address;       /* e-mail address                                     */
        int method;           /* how                                                */
        long delTime;         /* deliver time                                       */
        String filename;      /* file's name to store information                   */
        String port;          /*port number to send the message to particular port  */
        String msg;           /*message to be sent                                  */
}
```

The deliverStruct class is used by DCS services and applications to indicate to the Message Deliverer how to send the messages to the final users. The address value must be valid if the notification method is e-mail or z-write; the filename must be valid when the notification method is LOG_FILE.

5.5.2 User Interface

The user interface was developed using Java Swing. It consists of one main window and three sub windows.

1. **Main Window**

   This is the main DCS window. It has options for file, decision, notification, application etc. The user can choose one of the menu items and particularly for

notification the user has a choice of add, modify or delete a request. It will call sub windows to add, modify or delete the request. The main window has an announcement window on itself, which will display the instant message and will also display results of the queries asked by the user. Figure 5.1 shows the Main Window.

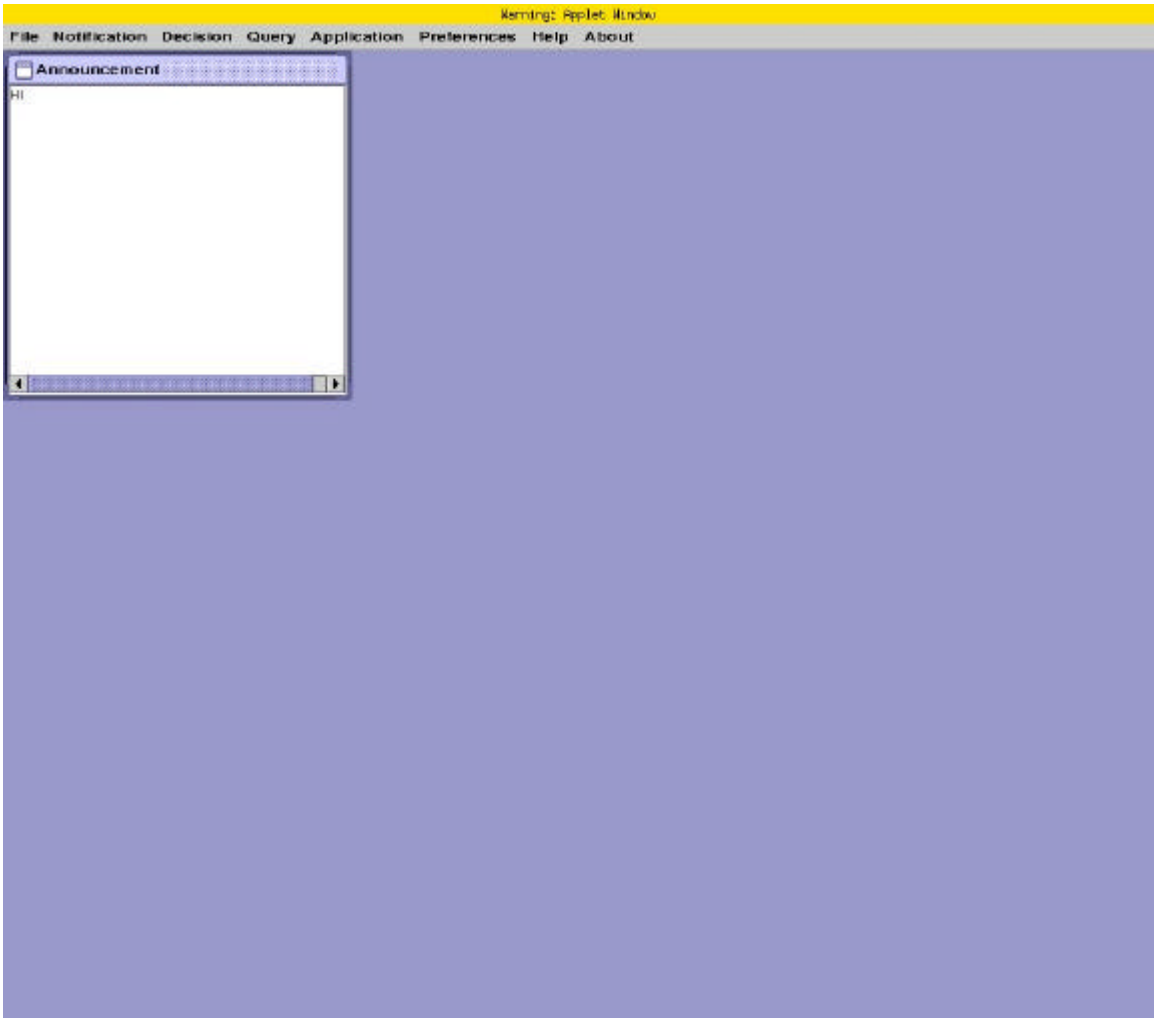

Figure 5.1 Main Window

2. Add Window

This window allows the user to select the different parameters for his request. The graphical user interface provides a list of events, methods of notification, formats, objects, members etc. Once the user makes his selection he can add it to the list of requests by pressing OK. Figure 5.2 and Figure 5.3 shows the Add Window.

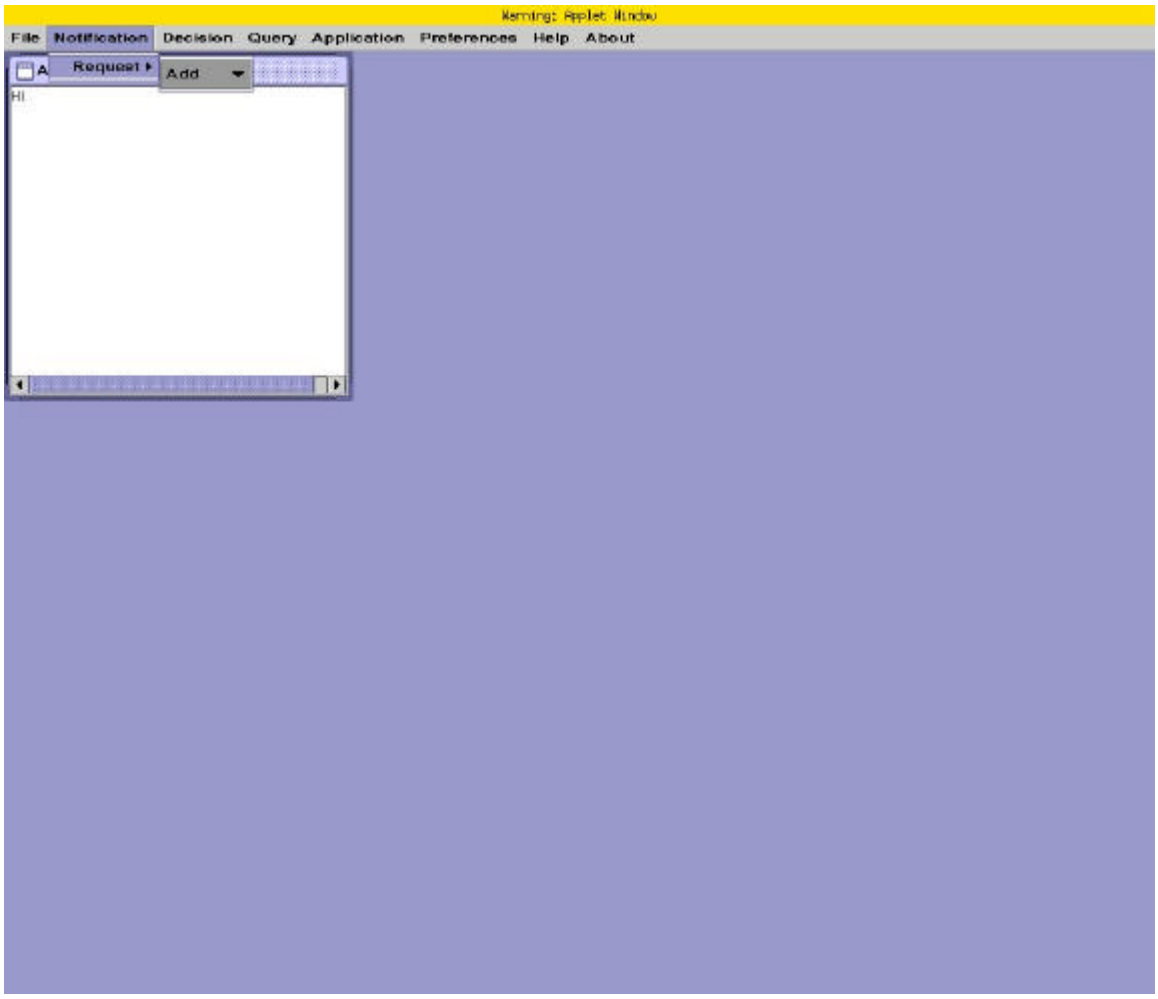

Figure 5.2 Add Window Screen 1

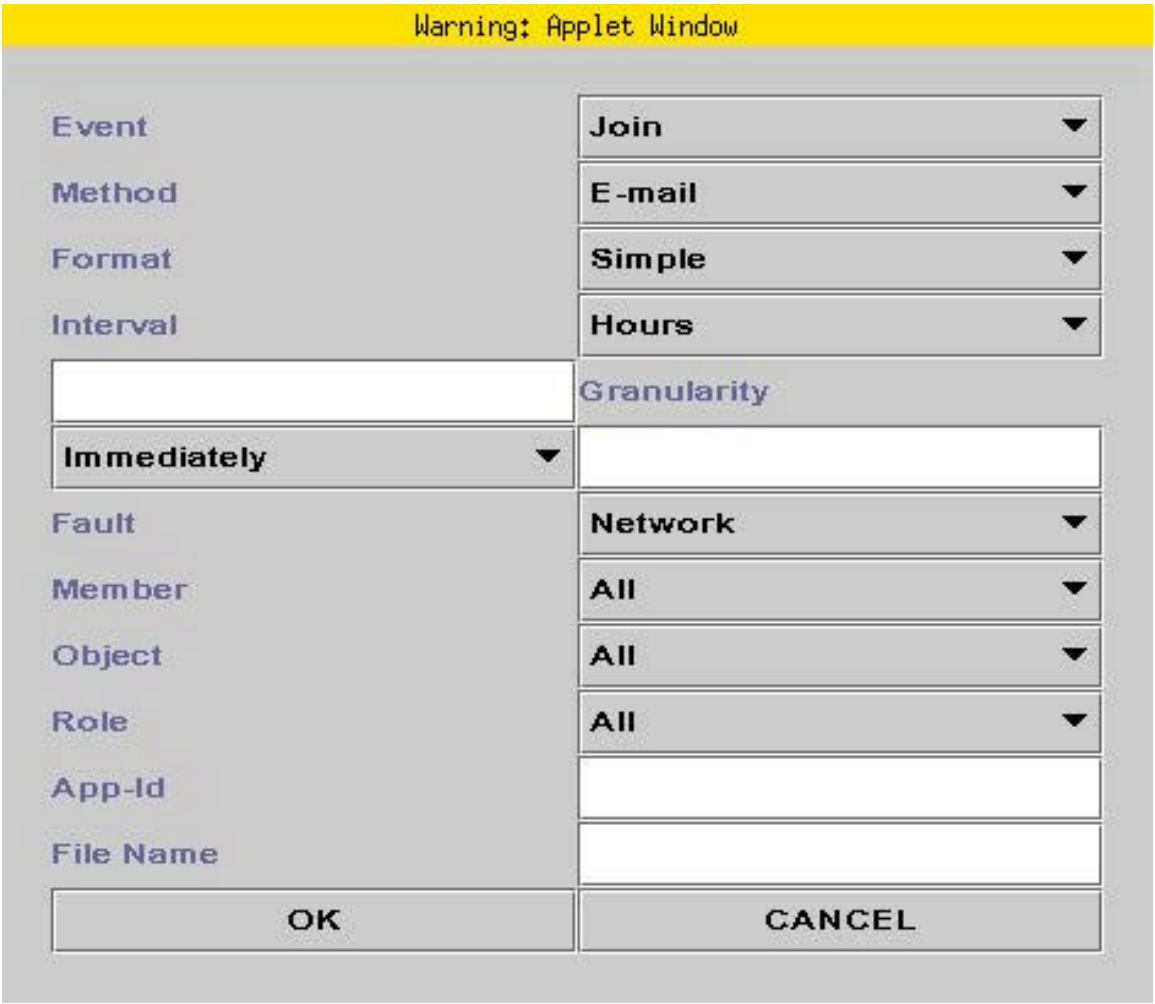

Figure 5.3 Add Window Screen 2

3. Modify Window

The user calls this window to make modifications to existing requests. The user will only be allowed to modify some of the attribute: time interval, time granularity, method and format, as explained in Chapter 4.

4. Delete Window

This window will display all the event of interest requested by this user. The user then can select the event of interest in which he is not interested anymore. This event will be deleted from the LNDB. Figure 5.4 shows the Delete Window.

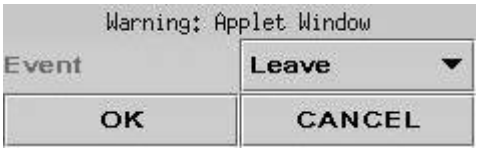

Figure 5.4 Delete Window

5.6 Testing

This section presents the evaluation process followed to check whether the system meets the requirements specified in Chapter 3. Two kinds of evaluations were performed: individual testing and integration testing.

For testing purpose, two more requests are added to the graphical user interface's Main Window. Delivery module and Raise event have been added to the Notification request menu.

5.6.1 Individual Testing

The purpose of this test is to check if NS correctly executes the different primitives. The modules were unit tested by combining the graphical user interface and the test programs that used different system primitives. Figure 5.5 shows the testing window.

Request Manager

We tested how the system behaves when different kinds of requests are made. All primitives were tested with different requests. Add, modify and delete are the choices

given to the user to add the selected event of interest, modify the selected event of interest and delete the selected event of interest.

When user selects “add”, the addwindow pops up which requires the user to provide the information about event of interest, method of notification, format of the message, time interval, time granularity and other things. After user adds the information, is gets stored into the LNDB.

When user selects “modify”, the modifywindow pops up which requires user to select one or more than one event of interest which they want to modify. They can only modify method of notification, format of the message, time interval and time granularity.

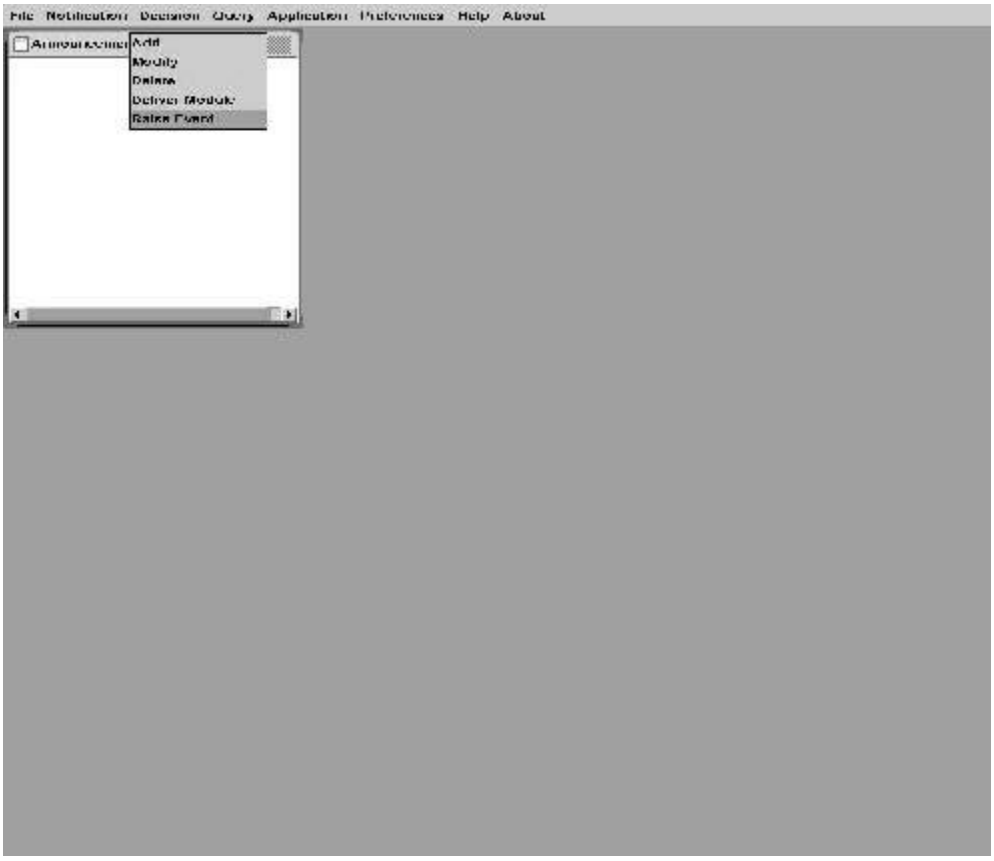

Figure 5.5 Main Testing Window

After user modifies the information, it gets propagated to LNDB where it gets modified.

When user selects “delete”, the deletewindow pops up which requires user to select the event which is not of his interest anymore. It displays all the requests made by user to give him the flexibility to choose one of them. After user deletes the information, it gets deleted from the LNDB.

Addwindow, modifywindow and deletewindow check if the required methods are called which perform addition, modification and deletion of the information.

Event Manager

To test the Event Manger, one more request was added to the Notification request menu, Raise event. Figure 5.6 shows the parameters required to enter there. This module calls the methods to create the message and also calls the delivery module if the request is immediate.

Raise event module checks to see if it calls the required method to create the messages and then appropriate method to deliver the messages. Event is raised with the required parameters. It calls ntfCreateMsg() method to create the message and then ntfNotify() method to deliver the messages.

Message Deliverer

To test the Message Deliverer, a deliver module request was added to the notification request menu. The user provides the required information about the message to be delivered, to whom (address) and by which method. Figure 5.7 shows the testing window of message deliverer.

Deliver module is the unit testing of message deliverer. It requires parameters according to the method of notification selected. For example, if the selected method of notification is email then it requires the message to be delivered and the email address.

It calls ntfNotify() method directly and this method inturn calls one of these methods: ntfEmail(), ntfZwrite(), ntfWindow(), ntfSelective(), ntfStoreF() and ntfPort().

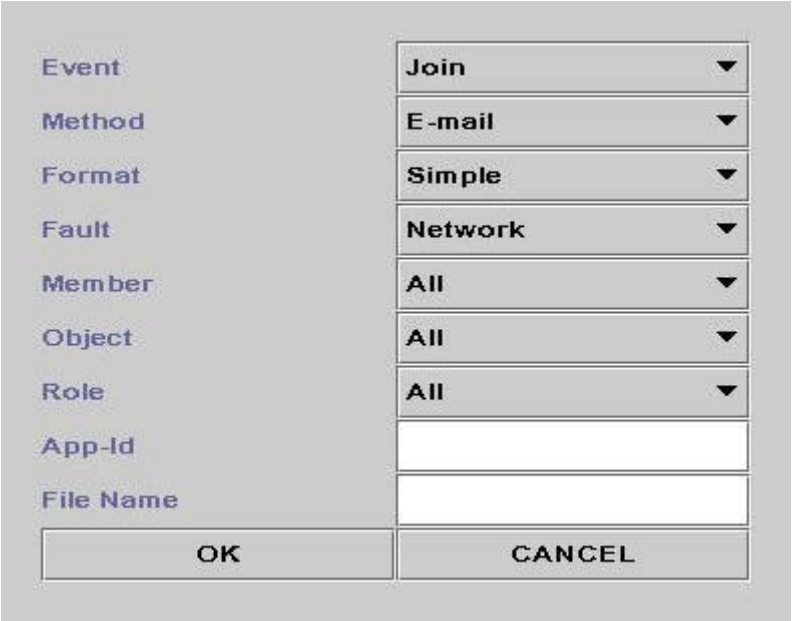

Figure 5.6 Raise Event Testing Window

Figure 5.7 Message Deliverer Window

5.6.2 Integration Testing

The integration tests check the correct operation of the system when it interacts with the different DCS's services. It is necessary to check that NS can access the information it requires from other services and that other services can access NS services.

The integration of Notification Services into DCS was made in different steps.

Client-server communication

The client sends the information provided by the user to the server. The server calls the appropriate methods depending upon the request from the client. Different tests will be performed to assure that the passing of parameters between the client and server be done correctly. Also, the same set of tests explained previously will be executed to check the correct functioning of the system.

DCS's Services

The second step will consist in the integration of NS with other DCS's services. This required us to include in NS the proper calls to the primitives offered by those services and that those services used the primitives provided by NS to ask for notification. At the time of this writing, these tests have not yet been performed since they require facilities not yet available.

5.7 Summary

This chapter discussed the implementation details for Notification Services in DCS v.2. Implementation details like classes used, message size, time manipulation, communications and program interface were discussed. Individual and integration testing

were also discussed here. The next chapter presents conclusions and potential enhancements to the system.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

The design and implementation of Notification Services for the University of Florida Distributed Conferencing System version 2 (DCS v.2) was successfully accomplished. The requirements set for this service were met by the design and implementation of the system.

The Notification Services provide a flexible and friendly tool for awareness of the actions taking place in the conference to the final users that interest them, even if they are not actively participating in the conference. NS allows users to select the events of interest, as well as message format, time constraints and notification method that best meet their requirements.

The databases supporting the operation of the system provide an efficient method to manage events that may happen in different administrative domains. The LNDB maintains the requests made by local users, providing a central responsible domain for managing the requests and creating the messages to be sent to those users. In contrast, the GNDB provides useful information that identifies the domains that should be informed about the events taking place around all administrative domains, reducing in this way the number of messages required to inform about event occurrences; only domains that are interested in an event receive information about it when it occurs. GNDB also allows efficiency as searches in the LNDB are reduced.

The different notification methods and the message deliverer permit NS to be used as a message deliver that can be employed by DCS users to communicate with one

another. The different DCS services now have the possibility of contacting the users. Notification Services can be seen not only as the mechanism for users to know about the events going on in the conference, but also as a mechanism for the different system's services to inform users and port about important events that may require users' attention. For example, the Fault Tolerance Service uses NS to inform users of faults that require human intervention.

6.1 Future Work

There are additional features that can be added to this system which were not included in this version that can be considered for inclusion in future versions of the system.

1. Additional notification message formats. The system provides two notification formats, but more sophisticated formats can be offered (e.g., summary reports that indicate the total number of times the events occurred in the conference). Moreover, a facility for the user to indicate the information required per event could give the user the flexibility to identify event information significant to him. For example, a user could be interested in being notified when a fault occurs in the system; he may only need to know the type of failure but not when it occurred. A Graphical User Interface can be used that can allow the users to select interested parameters. For example, if the user is interested in someone joining the conference, he can be given the associated parameters like member, role and time by the user interface and he can select the interested one. The message formats in this case will be:

   (a) Member _____ join the conference

(b) Member _____ join the conference with role _____

(c) Member _____ join the conference with role _____ at time _____

If the user selects only member then the message format will be (a). If the user selects member and role then the message format will be (b). If the user selects member, role and time then the message format will be (c).

2. Additional notification methods. Currently the system only supports some notification methods that may not always be available. The system should provide tools to configured the notification methods according to the available facilities. For example, if zwrite is not available another system could be used instead.
3. Configurable events. Notification services manage a fixed number of events. It would be convenient in some cases to have a configurable system that permits users to specify dynamically the types of events that need to be managed. This allows the system to be more adaptable to different applications environments.
4. System independence. Notification services need the cooperation of other systems and applications to know about events; it also uses facilities provided by other DCS services (i.e., Database Services and Conference Control). An independent system that could monitor the events and manage its own set of databases and communication facilities, would be a very convenient tool that could be used in different systems. Available Unix notification facilities could be useful to monitor the occurrence of certain events, for example, the active users in the system (who) and the status of processes (ps).
5. Instant message extension. If the user is not logged in to the DCS conference and if he has selected the method of notification as instant message, then he will not

receive the instant notification message. But in some cases, user might be interested in being receiving instant notification messages when he joins the conference again as a summary report which were not delivered before because he was not logged in to conference. This facility can be provided to the users.

6. Port extension. Sending a message to a port can be replaced with sending a message to a particular machine. Instead of providing port number, destination hostname or IP address can be provided. Also the format of the message to be sent could be in NOCOL, ASN.1 or XML.

6.2 Summary

This chapter presented the conclusions derived following the development of Notification Services in DCS v.2, as well as the potential enhancements that can be performed to the system, which include; constraints relaxation, additional notification message formats, additional notification methods, configurable events and system independence.

APPENDIX
APPLICATION PROGRAMMER INTERFACE NOTIFICATION SERVICES

The Notification Application Programmer Interface (API) has three primitives: ntfRequest(), ntfEvent() and ntfNotify().

1. **ntfRequest()** ntfRequest() adds, deletes or modifies a user's request. This primitive has three methods (addFunc(), modifyFunc() and deleteFunc()).

   int addFunc(String eventId, String format, String method, …)

   The user must indicate:

   - Event of interest
   - Format of the message (SIMPLE or DETAILED)
   - Method of message delivery (WINDOW, ZWRITE, EMAIL, SELECTIVE, LOG_FILE, PORT)
   - Time interval
   - Time granularity
   - Member name
   - Object name
   - Application id
   - Role of interest
   - Filename
   - Fault type

int modifyFunc(String method, String format, String interval, String granularity, int time_interval, int time_granularlity)

The user can modify:

- Format of the message
- Method of message delivery
- Time interval
- Time granularity

int deleteFunc(String eventId)

The user can delete:

- Event of interest/interests

Output: Returns 0 on success or error code on failure.

2. **ntfEvent()** This primitive is used by other DCS services to inform NS about the occurrence of events. It takes the information about an event that occurred in the conference and searches the LNDB for users interested on it. In case the event received is *loss role, loss membership* or *new role,* the affected user is notified by email about it. Moreover, if the event is loss role or loss membership, the invalidated requests are deleted from the database.

```
int ntfEvent(eventStruct ev, int location)
eventStruct{
        int eventId;
        String time1;    /* to indicate at what time ocurred the event     */
        int objectId;
        int dcsId;        /* dcsId of user that lost role, or has new email */
```

```
        int role;          /* to indicate what role was lost              */
        int fault;        /* to indicate faulty system, process?          */
        int faultType;
        int applId;
        int confId;
        String result;  /* to indicate result of voting, motion descrip. */
        String member;
        String rol;
        String object;
        String app;
        String address;
}
```

Input: event information

Output: Returns 0 on success or error code on failure

Calls: ntfDelete(), ntfImmediate(), ntfList()

3. **ntfNotify()** This function delivers a message to a user using the method specified.

```
int ntfNotify(String msg, deliverStruct del)
deliverStruct{
        int dcsId;          /* to whom                        */
        int method;       /* how        */
        long delTime;  /* deliver time                    */
        String filename;
        String address;
```

```
        String port;
        String msg;
}
```

Input: notification message, information to deliver the message.

Output: Returns 0 on success or error code on failure.

Calls: ntfEmail(), ntfZwrite(), ntfWindow(), ntfSelective(), ntfStoreF(), ntfPort().

LIST OF REFERENCES

[1] S. Greenberg. Computer-supported cooperative work and groupware: an introduction to the special issues. *International Journal of Man-Machine Studies,* 34:133-141, 1991.

[2] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware some issues and experiences. *Communications of the ACM,* 34(1):39-58, Jan. 1991.

[3] *The Random House Dictionary of the English Language.* Random House, New York, 1971.

[4] *Webster's Third New International Dictionary.* Merriam Company, Springfield Massachusetts, 1971.

[5] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada,* 107-114, October 1992.

[6] R.E. Newman-Wolfe, C.L. Ramirez, M. Montes H.K. Pelimuhandiram, M. Webb, and D.L. Wilson. A brief overview of the dcs distributed conferencing system. University of Florida Department of Computer and Information Sciences, Gainesville, 1991.

[7] D. G. Tatar, G. Foster, and D. G. Bobrow. Design for conversation: lessons from cognoter. *International Journal of Man-Machine Studies, 34:185-209,* 1991.

[8] K. Watabe, S. Sakata, K. Maeno, H. Fukuoka, and T. Ohmori. Distributed multiparty desktop conferencing system: Mermaid. In *Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, California,* 27-38, October 1990.

[9] R.E. Newman-Wolfe and H.K. Pelimuhandiram. Mace: A fine grained concurrent editor. In *Conference on Organizational Computing Systems, Atlanta, Georgia,* 240-54, November 1991.

[10] R.E. Newman-Wolfe, M. Webb, and M. Montes. Implicit locking in the ensemble concurrent object-oriented graphics editor. University of Florida Department Computer and Information Sciences, Gainesville, 1991.

[11] M.D. P. Leland, R. S. Fish, and R. E. Kraut. Collaborative document production using quilt. *In Proceedings of the Conference on Computer-Supported Cooperative Work, Portland, Oregon,* 206-215, Sept. 1988.

[12] D. Georgakopoulos and M. Hornick. An overview of workflow management: From process modeling to workflow automation infrastructure. Distributed and Parallel Databases, 3(2):119-153, April 1995.

[13] D.B. Davis. Software that makes you work flow, *Datamation*, 75-78, Apr. 1991.

[14] S.K. Sarin. Workflow and data management in InConcert. In *Proceedings of the Twelfth International Conference on Data Engineering, California, IEEE*, 497-499, 1996.

[15] S.K. Sarin, K.R. Abbot, and D.R. McCarthy. A process model and system for supporting collaborative work. In *Conference on Organizational Computing Systems, Atlanta, Georgia*, pages 213-224, 1991.

[16] W.K. Edwards. Session management for collaborative applications. In *Proceedings of the Conference on Computer-Supported Cooperative Work, Chapel Hill, North Carolina*, pages 323-330, 1994.

[17] S.M. Kaplan, W.J. Tolone, D. P. Bogia, and C. Bigoli. Flexible, active support for collaborative work with conversationbuilder. *In Conference on Computer-Supported Cooperative Work, Toronto, Canada*, pages 378-385, 1992.

[18] Y. Golberg, M. Safran, and E. Shapiro. Active mail-a framework for implementing groupware. In *Proceedings of the Conference on Computer-Supported Cooperative Work, Toronto, Canada*, pages 75-83, 1992.

[19] M. T. Rose and k. McCloghrie. *How to Manage Your Network Using SNMP*. Prentice Hall PTR, Upper Saddle River, NJ, 1995.

[20] D. T. Perkins. *RMON: Remote Monitoring of SNMP-Managed LANs.* Prentice Hall PTR, NJ, 1999.

[21] G. Held. *LAN Management with SNMP and RMON*. Wiley Computer Publishing, New York, 1996.

[22] D.R. McCarthy and U. Dayal. The architecture of an active database management system. In *Proceedings on the ACM SIGMOD Conference*, New York, pages 215-223, 1989.

[23] R. Ramarao. Directory services for a distributed conferencing system. Master's Thesis. CIS Department, University of Florida, Gainesville, 1991.

[24] Sun Microsystems, Inc., Palo Alto, CA, http://developer.java.sun.com/developer/onlinetraining/rmi/, July 2000.

BIOGRAPHICAL SKETCH

Swati Patanjali Shukla was born in Ahmedabad, India. She received her bachelor's degree in computer engineering from L.D. College of Engineering in Ahmedabad, in 1998. She joined the University of Florida to pursue her master's degree in the spring of 1999 and she will graduate in December of 2000. Her research interests include communication and computer networks.