Aston University

Some pages of this thesis may have been removed for copyright restrictions.

If you have discovered material in AURA which is unlawful e.g. breaches copyright, (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please read our Takedown Policy and contact the service immediately

SERVICES FOR ACTIVITIES IN GROUP EDITING

'SAGE'

BALA BUKSH

Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

July 1993

© The copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without proper acknowledgement.

THE UNIVERSITY OF ASTON IN BIRMINGHAM

SERVICES FOR ACTIVITIES IN GROUP EDITING

'SAGE'

BALA BUKSH

Doctor of Philosophy

1993

Summary

The project described in this thesis investigates the needs of a group of people working cooperatively in an OSI environment, and recommends tools and services to meet these needs. The project looks specifically at Services for Activities in Group Editing, and is identified as the 'SAGE' project.

The project uses case studies to identify user requirements and to determine common functionalities for a variety of group editing activities. A prototype is implemented in an X.400 environment to help refine user requirements, as a source of new ideas and to test the proposed functionalities.

The conceptual modelling follows current CCITT proposals, but a new classification of group activities is proposed: Informative, Objective and Supportive application groups. It is proposed that each of these application groups have their own Service Agent. Use of this classification allows the possibility of developing three sets of tools which will cover a wide range of group activities, rather than developing tools for individual activities. Group editing is considered to be in the Supportive application group.

A set of additional services and tools to support group editing are proposed in the context of the CCITT draft on group communication, X.gc. The proposed services and tools are mapped onto the X.400 series of recommendations, with the Abstract Service Definition of the operational objects defined, along with their associated component files, by extending the X.420 protocol functionality.

It is proposed that each of the Informative, Objective and Supportive application groups should be implemented as a modified X.420 inter-personal messaging system.

KEY WORDS: Services for Activities in Group Editing, Group Editing Services, Joint Editing Tools, OSI & Group Editing and Group Editing and Group Communication System.

Thank-God

Dedicated to My Loving Mother

Acknowledgements

I would like to express my sincere appreciation and thanks to the following persons:

Mr. Bernard Doherty, from Aston University, and Dr. Andrew Jordan, from the University of Huddersfield, who have made this research possible by their continous invaluable guidance, support and valuable time.

Mr. John Williams, Aston University, for offering valuable help and suggestions on 'quipu' where PP is implemented.

Dr. Steve Benford, Nottingham University, for providing needful guidance on the project and supplying various literature on the subject including CCITT draft and AMIGO report (a complementary copy).

The staff of the department and all those persons with whom personal communications have been made during the case studies and demonstration of the prototype.

The Government of India, which has sponsored this research programme and provided complete financial support.

The High Commission of India in United Kingdom, specially Mr. S. Mukharjee, Student Welfare Officer, who has met official requirements and arranged all payments.

The Oil & Natural Gas Commission of India, my empoyer, who has granted the study leave for completion of the study.

Finally, I am deeply grateful to my wife Sajan, for her sacrifice during my stay in the United Kingdom and my son Pankaj and daughters Priyanka and Renuka for their sacrifice and immense patience.

Table of Content

Chapter 4 : Group Communication Architecture and Models

Chapter 5 : Case Studies & Identification of General Model

Chapter 6 : Editing A Newsletter: Prototype Specification

Chapter 7 : Testing & Evaluation of Newsletter Prototype

Chapter 8 : Supporting Case Studies

Chapter 9 : Proposed Model for Group Editing Activities

Chapter 10 : Recommendation for Group Editing Services

Chapter 11 : Conclusion

List of Figures and Tables:

Chapter 1

Introduction

1.0 Introduction

The aim of this research is to investigate the needs of a group of people working co-operatively and communicating in an OSI network environment, and to recommend tools and services to meet these needs. The research focuses specifically on the services and tools required for group editing activities.

The research described in this thesis is concerned with **Services for Activities in Group Editing** (**SAGE**), and will be identified as the '**SAGE**' **project** for the remainder of the thesis.

Information technology has in recent years seen a move towards individual work-stations linked by networks. These networks have become widespread, allowing global communication for many users. The improved communication capability has been exploited in many ways. One of the recent developments has been computer supported cooperative working (Taylor, 1990). CSCW provides the computing environment that allows several people to work co-operatively on a common task. '*Computer supported cooperative work (CSCW) combines the understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services and techniques*' (Wilson, 91). CSCW promises to provide a timely synergism between group dynamics and state of the art computer technology (Cross, 89). The growth of CSCW has led to research towards the development of tools for coordinating group activities. These tools have been described

as group communication services for groupware, conferencing, team work or co-operative work. Some tools are available but do not fully address the need of particular groups of people working together over a distributed environment. What is needed is a system which allows propagation of information between a large number of people in an organised way, with users ideally not needing to be aware of the underlying services (for example e-mail, unix-mail, X.400, EDI or any other messaging services) used to handle the communication.

A variety of group communication applications has been developed. These applications include computer conferencing, bulletin board and news services systems (e.g. COM, USENET news and EIES and VAXNOTES). An example is a departmental bulletin board which is a special kind of mass communication arrangement for posting of announcement, messages and decisions written by, and intended for the users of the system. The applications are focused on providing shared access to messages within a group. These systems introduced many important ideas, but they suffer from a number of limitations (Benford et.al., 90) particularly in functionality management and interworking. Over recent years there has been a movement to introduce standardisation to improve interconnectivity and provide a common flexible infrastructure in a widespread communication network. This infrastructure is provided within the Open System Interconnection (OSI) environment.

The OSI standards cover most aspects of network communication. Within OSI the X.400 series of recommendations are designed to provide the standard protocols and message format that allows exchange of messages on a global basis. X.400 has been widely accepted, and covers the academic world, governments, and businesses. The 'SAGE' project seeks to define a common messaging protocol on a specific area of group working (i.e. group editing). The protocol is based on the X.400 series of recommendations, in particular the X.420 recommendation.

The 'SAGE' project classifies group activities into different application groups which are modelled as a group communication system and focuses on group communication support for group editing activities, an example of which is editing of a Newsletter. The 'SAGE' project seeks to identify services and tools for management, storage and distribution in support of group editing activities (so as to fulfil the requirements of the group of people working together). The work follows the direction proposed by the CCITT draft working document on group communication X.gc (Joint ISO/IEC/CCITT, 91) and also takes account of research work being carried out elsewhere. Figure 1.1 shows the proposed relation of group editing services to the X.400 recommendations. The X.gc recommendations propose a group communication framework model and define asynchronous computer conferencing on top of it. The 'SAGE' project proposes that a shell structure be used with all group communication activities built on the group communication framework model. Group editing is one of set of group communication activities.

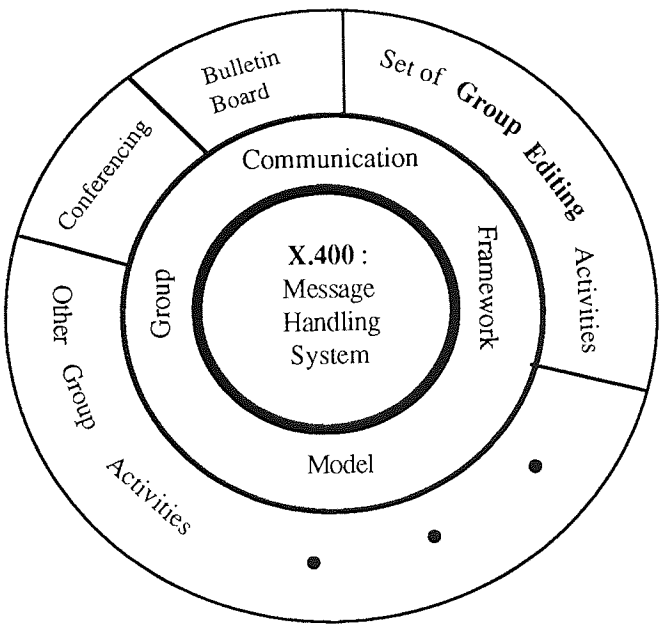


Figure 1.1 Relation of Group Editing Services to X.400.

The services needed in group editing are determined by a study of current literature in group communication and by interviewing a small set of potential users. From the services found to be required, a set of recommendations for tools to implement the services is described, in a form consistent with that used to describe other group communication support services (similar to the X.gc document).

To support the study, a prototype tool set has been implemented to provide the functions and services identified as needed to support group editing. This prototype has been implemented in the X.400 environment and used both to confirm that the service definitions of the tools are workable and to provide a testbed for user feedback which improves the information obtained from interacting users.

1.1 Value of Standards

The 'SAGE' project is concerned with productive communication over extensive networks. These networks are assumed to be international and to extend to all kinds of organisation. This can only occur if the services provided are in accord with widespread standards. International standards are thus crucial to successful provision of group editing services.

Information technology, in particular computer network technology, covers the converging disciplines of computing and communications and is thus a distributed technology involving equipment and systems from different suppliers. In such circumstances standards become essential to both the suppliers and the users. The organisations ECMA (European Computer Manufacturers Association), IEEE (Institute of Electrical and Electronics Engineers), CCITT (International Telephone and Telegraph Consultative Committee) and ISO (International Standards Organisation) have accordingly given due attention to standards (Bartlett, 1986). CCITT uses the term 'Recommendations' rather than standards.

Standards are needed when several parties participate in technical activities (Sherif & Sparrell, 92). Examples of such activities are: propagation of information, management of systems, specification of performance and quality attributes and compatibility and interconnectivity between applications. Standards facilitate the relationships among network vendors, users and providers. Open System Interconnection (OSI) brings together such parties, anticipating the availability of technology.

1.2 Network Protocols and OSI Networking

The major advantages of OSI standards have been discussed in last paragraph. In the light of these advantages, this research has been carried out within the OSI standards. The OSI standards define common protocols for communication in an open environment to meet the needs of different users.

In order for two processes to communicate over a communication link they need to implement various 'protocols'. The protocol definition in an 'analysis' of the OSI model of the open system interconnecting (Strokes (ed), 86) is described in the following terms:

> *"When we have two processes facing each other across some communication link, the protocol is the set of their agreements on the format and relative timing of messages to be exchanged. When we speak of a protocol, there is usually an important goal to be fulfilled. Although any set of agreements between cooperating (i.e. communicating) processes is a protocol, the protocols of interest are those which are constructed for general application by a large population of processes in solving a large class of problems".*

OSI standards solve these information management problems and address the communication needs. The following four aspects of functionality must be considered in designing a dynamic multi vendor network: Connectivity, Interoperability, Manageability and Distributed Applications. These functionality aspects are mapped onto the OSI reference model.

1.3 OSI Reference Model

The Open System Interconnection Reference Model (OSIRM) (Strokes (ed.), 86) is a seven layer model of a network 'architecture', with each layer performing a different function and having its interface to adjacent layers fully specified. The model defines the accepted International Standards by which open systems should communicate with each other, and also sets the rules for how the standards are to be implemented. An open system is a computing system or network that can interoperate with any other open system on the basis of OSI International Standards.

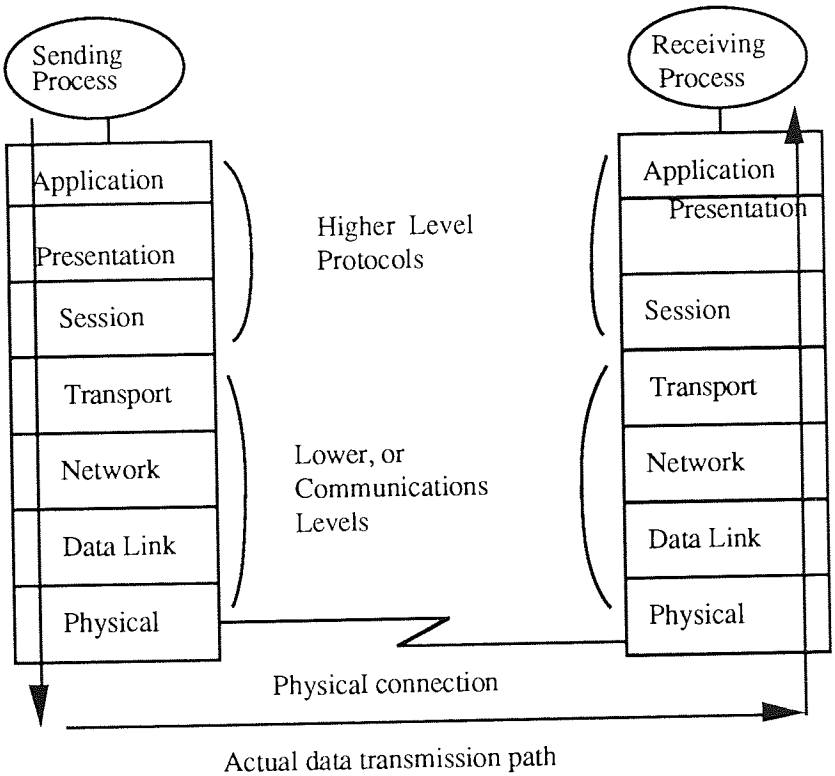


Figure 1.2 Seven layer architecture of OSI Reference Model.

The term 'architecture' is used to describe how a network organises connectivity and interoperability so that elements of a communication function in a logical order as information moves in and out of the network. A network's architecture is important because it sorts communication functions into logical groups, or layers, so that they

always perform the required function (Jarratt, 1989). Figure 1.2 shows the seven layer architecture of the OSI reference model (Taylor, 1986).

The functionality aspects (viz Connectivity, Interoperability, Manageability and Distributed Applications) are mapped on OSI reference model in a seven layer structure. **Connectivity** is the ability of the network to move any piece of information from one point to another, regardless of the medium or transmission technologies. Connectivity provides information integrity and ability to make physical connections. **Interoperability** is the ability of all system elements to exchange information between equipment from the same vendor or a collection of different vendors. It is this feature that makes information understandable. **Manageability** is provided by the combined products and services that enables the network to be designed, controlled, directed and supported, and that enable the management of change in a responsible and flexible manner. **Distributed Applications** enhance the usefulness, or value delivered by the network, providing the users with enterprise-wide, easy-to-use communication capabilities (Jarratt, 89).

Every layer in the OSIRM is defined by a standard which comprises a Service Definition and a Protocol Specification. Each of OSI's layers provides services to the layers adjacent to it, and maintains a relationship with the corresponding layer of the system it is communicating with. The layers are independent of each other in the sense that a change in one layer does not affect operations or protocols in neighbouring layers. For example, changing the physical transmission medium (e.g. Co-axial, Twisted pair, and Fibre links) does not affect the type of data link. The OSI Physical, Data-Link and Network (1, 2 and 3) layers address the connectivity needs of Local Area and Wide Area Networks (LAN and WAN). Interoperability aspects of the network are addressed by the next three layers: Transport, Session and Presentation (4, 5 and 6). Distributed Applications issues are addressed by the seventh, the highest layer. All seven layers of

the model are involved in providing manageability aspects of the network. A functional overview of each layer is discussed in Appendix 1.

An electronic message handling system should provide message input assistance, message transfer, status reporting, type conversion, and format preservation (Palme, 1987). Different vendor computer based mail systems use different terminals, formatting, addressing and control procedures. Recent efforts by CCITT have provided standard document formats, electronic mail and messaging and addressing structures as a basis for message handling. CCITT's protocols provide a means of harmonising these different paths. The **Message Handling System** (MHS) is defined by a set of standards for electronic mail. The standards are the **X.400 series** of CCITT recommendations and have been adopted by ISO as the Message Oriented Text Interchange System (MOTIS).

Two message services, Message Transfer (MT) service (a general, i.e. application independent, store-and-forward message transfer service) and Inter-Personal Messaging (IPM) service (person-to person communication), are described in the X.400 series of recommendations, together with their functional models and protocols. The protocols of the X.400 series of recommendations extend the Open System Interconnection environment up to the user level and thus provide rapid and effective transfer of information (in required format and shape), rather than data, by allowing automation of many of the information handling and routing functions (Roechr, 1986).

The X.400 recommendations provide a standard protocol that allow the exchange of information on a global basis. Implementations of the X.400 recommendations are widely available. Thus using the X.400 protocols as a carrier to derive group communication system services would increase the force of implementation of these services as standard on user community.

1.4 The 'SAGE' Project and the OSI Model

The Application Layer is the highest layer in the ISO model and it can be considered as the layer in which all application processes reside. The Application Layer provides information services to support end-user application tasks such as file transfer, remote file access database management, electronic mail and terminal access. It manages communications between applications.

The work of the 'SAGE' project is in the application layer, and is particularly concerned with the X.400 (i.e. message handling system) series of recommendations. The position of the 'SAGE' project research within the application layer is shown in figure 1.3.

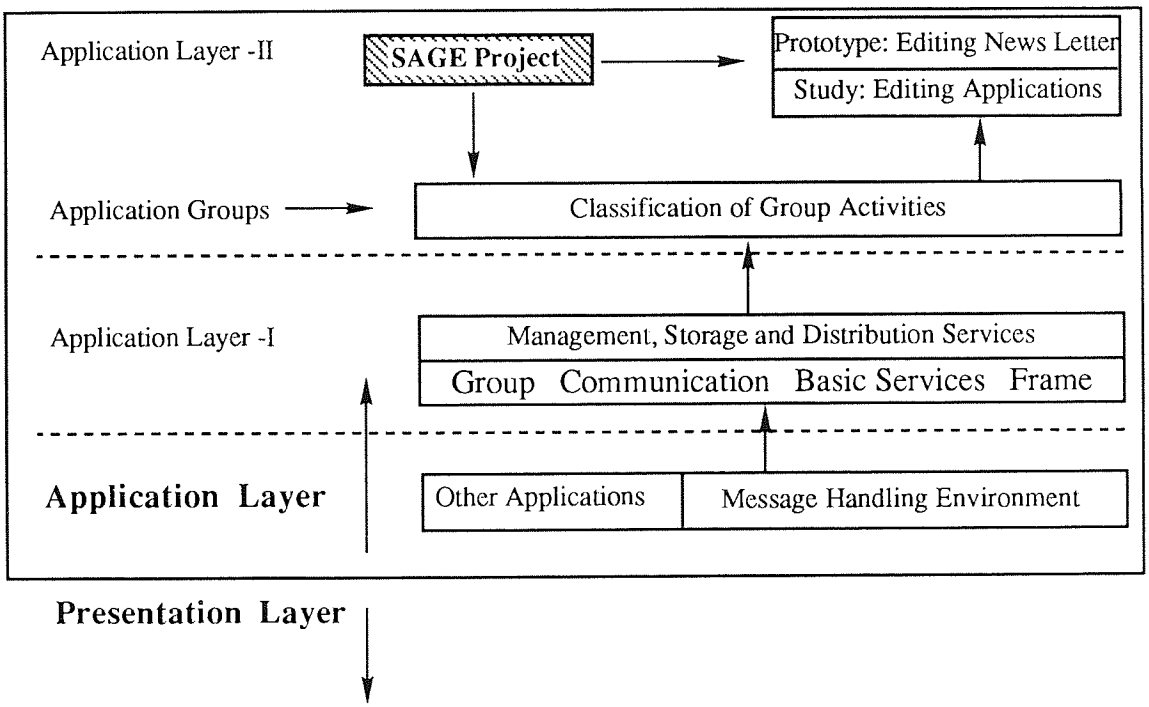


Figure 1.3 Extended Structure of the Application Layer.

1.4.1 The 'SAGE' Project and the Application Layer

X.400 and X.500 (directory services) recommendations enable interpersonal electronic mail and distribution list in a geographical distributed environment. The services of

X.400 recommendations (i.e. message handling system) form a message handling environment, and exist within the application layer. The X.500 (directory) services are also a part of the services provided by the application layer as a standard. Both X.400 and X.500 services do not support group activities such as cooperative writing, editing of periodical newsletters, meeting scheduling systems, which are based on the messaging system.

The 'SAGE' project defines Group Communication as "communication which fulfils the communicating requirement for a group of people with common objectives working on a single pool of information within the boundaries of the group". The complexity of the group, however, may depend on the group environment (e.g. members of a group or the group itself can be a part of other groups). The objective of the group may vary according to the nature of the problem in business, government and organisations. Group Communication sets up an environment for communication between more than two users[1] depending upon the group category (discussed later).

The X.400 recommendations have been used for user applications such as InterPersonal Messaging, Electronic Data Interchange, and are now the basis for work (AMIGO and X.gc draft) to define group communication tools (i.e. Group Communication Services) for group applications over networks using X.400 recommendations. The realisation of these services is shown in figure 1.3, in the Application-I layer, as 'group communication basic services frame'.

1.4.2 Outline of the 'SAGE' Project

The 'SAGE' project defines a classification of group activities into different application groups and focuses on the services and tools required for group editing activities within one application group (shown in figure 1.3). The aim of this research is to investigate

[1] A process can be user.

the needs of a group of people communicating together and to recommend tools and services to fulfil such requirements in an editing environment. Examples of user services and tools needed are: a mechanism to handle the information; setting up the environment for version and access control; structured storage; the header components (in context of X.400) and the need to manage the activity with operations such as create, update, add and delete.

The 'SAGE' project follows recent suggestions (Benford & Palme, 93) and takes a top-down approach to the development of group communication system services by ascertaining needs in group communication applications. The research is carried out in three phases.

The first phase involves carrying out case studies considering three group editing activities, viz a Newsletter, Technical Paper and Software Development Team. The Newsletter activity is discussed in depth and used for the prototype. The other two activities are discussed as supporting (minor) cases, with consideration of their similarities and differences. The case studies methodology seeks to determine the underlying needs for these activities by reviewing the literature and by interviewing some potential users involved in group activities.

The second phase of the research involves development of a working prototype for editing a Newsletter. The case studies and literature are the basis of the requirement specification produced to enable the design of the basic system for the prototype.

The third phase of the 'SAGE' project involves modelling of group editing activities based on common issues determined from the previous phase, and recommends the services and tools required for group editing activities in the light of the X.400 series of

recommendations. The recommended services and tools are proposed to modify/extend the functionality of the X.420 recommendation (Interpersonal Messaging System).

These phases of the research established that the main issues are to provide version and access control, to work out the logical structure of the common store, to construct the group editing header components and to provide management of the activity. These issues are discussed later in detail.

1.4.3 Research Method

The approach to research for the 'SAGE' project involves the following activities:

i) Carrying out case studies for three group editing activities to derive common functionality,

ii) Prototyping one of the activities, i.e. editing a Newsletter, as a testbed to users and to test the functionality,

iii) Deriving a general model for group editing,

iv) Identifying additional services and tools required for group editing activities,

v) Defining Abstract Service Definitions for additional operations required to meet the functionality of the group for editing activities,

vi) Mapping the services onto the Message Handling System as extension services of MHS,

vii) Proposing extended header parameters and their component files.

viii) Review of group activities in the light of classification of group activities,

ix) Proposing identification of these classified application groups as OSI applications as standards.

1.4.3.1 The case Studies

To understand the nature of problem and to produce an abstract specification, three group editing activities were chosen for case study. The initial 'finding' phase comprised arranging interviews with users who are involved in editing activities. The users are not concerned with the underlying services used for the system. However they are concerned with the issues like: avoiding redundant editing, the need to have more than one version, control over the activity, access facilities, the role of the referee in the case of technical articles and having header parameters (e.g. up-dated on, submission date and subject etc.). The role of group editing organiser (who acts as moderator) provides different kinds of control in each of the cases (discussed later). The case studies were useful in identifying the commonality of editing issues so that the necessary services for group communication, viz creation of group, definition of activity and creation of common storage, can be determined. These issues are key implementation issues in the context of creating a group editing activity before operation in a distributed network environment.

1.4.3.2 The Prototype

To support the development of ideas and investigate the required functionality, a working prototype has been developed for editing a Newsletter activity. The prototype system is divided into two sub-systems. One system contains the built-in processes, which act on behalf of the group editing organiser or group member concerned and take an automatic decision (discussed in detail in chapter 6) depending upon the event. This is a rule-based system. The built-in processes address issues such as version control, request handling to avoid redundant editing, delivery failure, reminder generation and automatic generation of a Newsletter document.

The second sub-system requires interaction with the group member concerned to act on an event. Amongst the tools provided are: definition of the Newsletter, addition and deletion of members, showing the differences between two contributions submitted by a member and showing the history and statistics of the activity. This set of facilities is presented as a menu for the convenience of users (for group editing organiser and other members).

The prototype uses the 'C' language, the Unix shell and the services of an X.400 implementation in the message handling environment. Finally the prototype has been demonstrated to potential users before reviewing the case and reaching conclusions and providing recommendations on additions and modifications to X.400.

1.4.3.3 Review of Group Activities

There are many kinds of group activity, which are discussed more fully in chapter 4. It may not be possible to have universal tools for each and every activity. In order to recognise services and tools for a group activity as an OSI application, there is a need to categorise/classify such activities. The classification of the activities is based on the behaviour and functionalities of the activities. For example the activities which supply information, such as a bulletin board, and activities which need other member's co-operation to complete a task, such as co-operative writing, should be in different categories. Based on the characteristics of activities (those which need similar kinds of support), it is suggested in this research that such group activities may be divided into three base classes of application groups (described later in detail). These classes or application groups are proposed as:

Informative group (supplies information only),

Objective (decision making) group (needs decision making support),

Supportive (co-operative work) group (co-operation is necessary to complete task).

This chapter has described the value of standards in a computer network environment along with the role of network protocols. An overview of OSI reference model has been discussed in relation to the 'SAGE' project, particularly the application layer and message handling system. Finally an outline of the research method has been described.

Chapter 2

Literature Review

2.0 Introduction

The literature review is structured to present a review of general work on group communication applications and then a review of specific projects such as 'AMIGO', 'GRACE' and by CCITT working group. Some current work on group communications is also described in this chapter. Literature that is specialised to a particular topic is reviewed in the chapter that covers that topic.

The existence of message handling system within the OSI reference model has been described in the previous chapter.

The literature shows that, in recent years, there has been a widespread growth of message handling systems and an increasing use of group applications over them. Research on group communication activities is underway in most countries and in many supplier organisations. Review of the literature (Wilson, 91) suggests there are about 50-60 work centres involved in developing group communication services. The extensive research in the field is difficult to completely review in this short space.

The literature discussed here is that which has made a significant contribution to the field and led to work related to the 'SAGE' project. More specific issues and aspects are discussed and referred to at appropriate places in later chapters.

2.1 General Work on Group Communications

A series of papers (Palme, 1986, 1987, 88) describes group communications. The first paper sets out the difference between personal and group messages. The paper also discusses information overload in personal communication and use of distribution list to reduce such overload. The second paper defines Computer Conferencing and its purpose. The paper describes conferencing and bulletin boards as group communication messaging systems with special support. The paper also describes the advantages (to users) of computer conferencing over face-to face meeting, and lists the common features of a conferencing system such as structuring of messages, access control (for open, closed or protected conferences), moderator functions and conversations. The paper then describes the principle of conferencing in the light of extension to the message handling system and X.500. The role of the moderator in group communication is first set out in the paper by Palme. The last (third) paper extends the ideas of the second paper on computer conferencing. The paper describes the detailed functionality of computer conferencing system in the light of X.400. The paper highlights the database structure for user and conference and the properties of conferences (open, closed and restricted). The role of X.500 (i.e. the Directory services) is extended in the conferencing system. The paper describes the operations for moderator and recipients and the use of distribution lists. The paper then gives a view on computer conferencing services in distributed systems in the light of CCITT recommendations and ISO standards. Operations such as voting and features of joint editing and database cleaning are also mentioned in the paper. Some of the features described by Palme appear later in CCITT draft X.gc (Joint ISO/IEC/CCITT, 91).

Two papers (Lanceros & Saras, 88, 89) introduce a model for group communication facilities in the X.400 message handling environment. The papers describe storage, directory and management services over message handling system. The papers give a

substantial description of a structure of a 'group' in the context of group communication. A group as described by Lanceros & Saras consists of group name, group members, distribution of contributions, common store, description of the group and the owner/ administrator. The papers consider group contributions as extension of the interpersonal messaging (IPM) format. The papers also describe the possibility of managing group messages by modification or extension to the protocols "P1", "P2" (P1 is a message transfer agent to message transfer agent protocol and P2 is an interpersonal messaging protocol) and directory services. The concept of entity group manager (similar to moderator in Palme) is supported in the papers. The work of Lanceros & Saras is in the context of the AMIGO MHS$^+$ project (Smith et.al., 89). Some of the features described by Lanceros & Saras appear later in CCITT draft release X.gc (Joint ISO/IEC/CCITT, 91).

The **AMIGO MHS$^+$** project has carried out research to explore Group Communication in the context of a distributed environment in the light of extension and integration of existing inter-personal services such as X.400 and Electronic Mail. Another concern is to close the gap between the facilities offered by existing centralised Group Communication services (e.g. conferencing systems and bulletin boards) and standardisation efforts in distributed messaging (X.400). The AMIGO MHS$^+$ report (Smith et.al., 89) discusses a distributed group communication service architecture and describes an abstract model for group communication support and an outline implementation architecture (described in section 2.2.1).

A working document X.gc (Joint ISO/IEC/CCITT, 91) on Group Communications has been defined by ISO/IEC/CCITT as "the overall system and services overview of Group Communications". This is the seventh version of the working document X.gc. It takes account of all previous work to derive the draft working document. The document defines a general Group Communication Model in terms of its goal, and requirements

and services to be provided. The working document also specifies group communication as an information model and then defines Abstract Service Definition of the system. The goal of this model is to provide a framework for modelling and implementing a variety of group communication applications (discussed in section 2.2.2). The 'SAGE' project accepts the proposed framework as a basis for research.

A paper by Jakobs (Jakobs, 91) takes a different view, with the assumption that the group communication may be divided into two distinct problem areas. These are known as user-oriented and network-oriented problem areas. The user oriented problems includes user-friendly naming techniques and establishment and maintenance of the groups. The network oriented problems are the mechanism to provide an addressing scheme and routing strategies. These include management of group addressing and multi-destination routing. The paper describes the integration of two services, viz message handling service and directory service, to support group communication. The paper then defines a new object class set for address and group description as Abstract Service Notation one (ASN.1) in directory services as a global naming structure. The paper also classifies the group into three categories: static, semi-static and dynamic and describes the techniques to define the addresses and group objects.

Jakobs' view differs from the group category approach used in the 'SAGE' project and discussed later in detail (in chapter 4). Jakobs' approach makes it difficult to consider issues like group classification and group communication problems in the context of group communication services and tools. Jakobs' proposal indicates that the group activity problems (other than user-oriented) are network problems. The paper does not make it clear that the network problems are to be built-in the network itself, or whether additional support should be provided to handle such problems.

A paper (Benford et.al., 92) describes the current state of art in group communication in terms of standardization work within the joint ISO/IEC/CCITT messaging group (X.400). The work has been discussed in the light of providing standardized support for 'Asynchronous Group Communication' over OSI network. The group communication is described as a general computer mediated communication (CMC) which is to be extended to Computer Supported Co-operative Work (CSCW), computer conferencing and group decision support systems. The paper discusses the importance of standards and supports the CCITT working document X.gc (Joint ISO/IEC/CCITT, 91).

The paper describes the scope of standardisation in terms of a framework reference model for group communication. As a reference model it then discusses the group communication framework in one part. The scope of a reference model for group communication include the modelling of group communication framework as a standard *Information model*. The second part of the paper describes the use of an Information Model to model the core functionality for asynchronous computer conferencing. The informal description of asynchronous computer conferencing in given along with the modelling of the operations with the information model. The paper raises important current issues.

Another paper on standards for OSI group communication (Benford & Palme, 93) provides an overview of the current working document X.gc (Joint ISO/IEC/CCITT, 91) as a basis for messaging standardisation. The paper describes the group communication architecture in the light of a distributed environment. It highlights the critical limitations of messaging systems, which are lack of persistent information (disappearance of messages i.e. not reaching to a destination) and lack of structure (retrieval of referenced messages) of information. The paper lists the successful group communication systems and the limitations of their being a non-standard products. The

paper then supports the view that for such services to be successful they should be built on the existing OSI standards. For example the building of group communication services on message handling system. The paper describes 'bottom-up' and 'top-down' approaches in context of producing group communication services as standard.

The paper by Benford and Palme splits the working document into two parts: specific group communication services information model and asynchronous computer conferencing. The asynchronous **computer conferencing** consists of a **set of group activities**. The paper describes several categories of membership (viz regular member, observer, prospective member, moderator and activity manager) as its participant. The paper supports the categories of group activities type (Palme, 87) as open, closed, restricted and protected. The group communication information model is object-based for all information in the group communication system. The abstract service definition defines basic operations for manipulating group communication system objects. The information model represents information and the members of the group as objects. The paper further highlights the operations of CCITT working document X.gc (Joint ISO/IEC/CCITT, 91) in reference to the base model. A mapping has been carried out of a computer conferencing onto the group communication reference model services. The paper also indicates features of the group communication system to support it, as an extended structure. These are support of global name space as an extension of directory service and the use of 'links' to build simple objects into complex information structures. The 'SAGE' project makes use of ideas in the paper and builds on the framework.

2.2 Specific Projects on Group Communication

The previous section describes work on group communication by individuals. There are some important projects which have contributed towards the standardisation process for

group communication. This section describes the AMIGO and the GRACE project and the associated CCITT draft proposal. The last section describes the scope of the 'SAGE' project in relation to the work discussed here.

2.2.1 The AMIGO Project

A report "Distributed Group Communication - the AMIGO information model" (Smith et al, 1989) describes the AMIGO MHS$^+$ project. The goal of this project is to explore Group Communication in the context of a distributed environment: it considers the extension and integration of existing inter-personal services such as X.400 and Electronic Mail. Another concern is to close the gap between the facilities offered by existing centralised Group Communication services (e.g. conferencing systems and bulletin boards) and standardisation efforts in distributed messaging (X.400). The report discusses a distributed group communication service architecture and describes an abstract model for group communication support and an outline implementation architecture.

The AMIGO project has three different work areas; Advanced Group Communication, MHS$^+$ (Message Handling System) and MMConf (Multi-Media Conferencing). The Advanced group communication area addressed the provision of a conceptual framework for group communication activities. The MHS$^+$ group was concerned with how to use the available OSI services for group communication support. The MMConf group examined on-line group communication and the requirements of multimedia tools and support services. AMIGO MHS$^+$ project recommendations have been used in most subsequent work on group communications including the CCITT draft document X.gc (Joint ISO/IEC/CCITT 91). The AMIGO MHS$^+$ project report is presented in various chapters, most of which have been influential in the 'SAGE' project.

An introduction to group communication service requirements is described by Smith (Smith, 89) which indicates the example of existing structured communication applications and defines the group communication activities. The ideas are followed while describing distributed architecture for group communication along with its elements. The work is recognised by the CCITT group in the draft X.gc (Joint ISO/IEC/CCITT, 91).

A contribution by (Benford, 89) in the AMIGO report specifies an information model for group communication as global information space. Benford examines the issues required for naming the communication entities and information objects. The environmental view of information objects, their operations and the access rights are also discussed and appear in the CCITT draft.

The AMIGO contribution by Weiss & Bogen, 89 describes the group communication architectural requirement from the view of users, administrator/providers and designers. Weiss & Bogen treat the group communication system services as basic (distribution, storage and co-ordination services) and advanced services. The contribution also describes the basic services of distribution, storage and co-ordination and the entity group communication service agent (GCSA) needed to fulfil service requirements. The chapter finally models the group communication system with the supporting services (viz message handling system, directory services, archive services and management services), along with the role of group communication service agent. These ideas appear in the 'SAGE' project.

One chapter (Nunez, 89) of the report gives an overview of archive services along with the storage protocols. Nunez describes a distributed Archive service agent used to handle information objects and operations. A concept similar to an archive agent has been used in the 'SAGE' project, but with extended functionality for advanced services.

The contribution by Wagner & Palme (Wagner & Palme, 89) in the report discusses additional group communication services to support added functionality that meets the requirement of advanced group communication. The chapter also describes a need for knowledge based information handling support in such functionalities. Wagner & Palme examine the use of knowledge based information handling system in archive services, retrieval, storage and distribution of information objects. This portion of the report identifies the activities which require advanced group communication support, viz voting, joint editing and meeting scheduling. The 'SAGE' project follows the idea of a knowledge-based information handling system, which is termed a rule-based information handling system, and focuses on joint editing activities.

2.2.2 CCITT Draft Release X.gc

The recent draft report by ISO/IEC/CCITT (Joint ISO/IEC/CCITT, 91) has defined "the overall system and services overview of Group Communications" in a working document. The draft is based on past work carried out in the field, including AMIGO and GRACE (section 2.2.3) projects, and input from the CCITT study group on group communications. This is the seventh version of working document X.gc. The document defines a general Group Communication Model in terms of its goal, requirements and services provided. The working document also specifies group communication as an information model and then gives an Abstract Service Definition of the system. The goal of this model is to provide a framework for modelling and implementing a variety of group communication applications. The document also defines basic terms in group communications such as: object, item, entity, domain, attribute, link and distinguished name.

The draft document specifies a general model of group communication and Abstract Service Definitions and models for asynchronous Computer Conferencing. The working document also points out some of the further study areas, for example: Access

Control mechanism, detailed functionality of filters (exploring links between objects) and alias support for objects. The document does not include however certain advanced Group Communication Services, for example, Voting, Joint Editing of a document and specific group activities.

The last part of the draft describes distributed relations of the computer conferencing model. The draft indicates the requirement should be meet by a special computer conferencing protocols by mapping of the computer conferencing services onto the general group communication model. The draft working document describes the role of group communication service agent (GCSA) to control the activity. The role of GCSA is defined in a similar manner to the message transfer agent in the MHS. Likewise each GCSA controls one or more activities and communicates with each other to meet the functionality of the activity. The working document also describes distribution of contributions in the group. The document finally describes the mapping of asynchronous computer conferencing into the basic architecture model in different ways. The draft gives warning that there will be a number of versions of this document, which is incomplete and should not be considered as a base for any implementations.

2.2.3 The Grace Project

The GRACE[1] project was a two year project funded by UK Joint Network Team to investigate and develop group communication applications within OSI network environment. The aim of the project was to specify and prototype an OSI-based group communication service which provides common support services for a wide range of group activities. This includes development of theoretical models of collaborative working and production of a demonstrator for one or more sample applications including computer conferencing. The phase I report (Benford et al., 1990) presents the

[1] The Group Communication project at Nottingham University is funded by UK Computer Board.

specification of the GRACE Conceptual Model as well as four examples of how the model would be used to represent different applications. The report identifies some activities to be modelled as GRACE which are described here in brief.

The GRACE conceptual model for group communication is an information sharing system within the groups. The model specifies a hierarchy of generic group communication objects and a set of abstract operations for manipulating them. The report indicates a list of group activities such as news distribution, joint editing, voting, elections, opinion polls, presentations and seminars etc. The report also describes the support requirement for group communication services in an OSI network environment. The report then considers the information overload problem (Palme, 86) along with other limitations. The report also describes the approach of the GRACE project to investigating the provision of an OSI based group communication services in two dimensions: the scale of communication (in terms of number of entities sharing information) and the degree of formalised communication procedures/constraints. The degree of formalised communication here refers to whether the activity involves a well-defined sequence of synchronised message exchanges (like formalised office procedures) or whether information is shared in a less synchronised manner (e.g. bulletin board). A GRACE conceptual model is also defined in the report, consisting of an information model and abstract operations required to derive the information in the model.

The GRACE report also describes the Formal specification of a GRACE group communication service and Schema definitions for the GRACE Generic objects. The definition of the specification uses the standard OSI Abstract Service Definition conventions. The conceptual model defines the functionality of the group communication system which provides the common support facilities to OSI group

applications. The document also specifies the schema for the generic item, cluster, role, domain and entity objects defined by GRACE conceptual model.

The sample of applications follows the common structure along with the overall functionality. Application specific objects and operations are mapped onto the conceptual model to derive the functionality. OSI networking is assumed and the project sets out to demonstrate conferencing and help desk applications over the OSI Group Communication Services. The ideas of GRACE project were useful to the 'SAGE' project , especially in mapping the services onto the message handling system.

Four activities considered as examples are: A Help Desk inquiry, Classified Advertising, "Which ?" style magazine and USENET news distribution service. The activities are of Informative type which are relevant to a specific group in terms of classification of group activities (discussed later).

2.3 Current Work in Group Communication

Group Communication is an active topic and work is being carried out world-wide. Some of the known major projects in the field of group communication are listed here. The 'COST 14 CO-TECH[2] action' is investigating the process of group knowledge development and assessing how technology could be made to support it, 'EUROCOOP[3]' and 'EIES 2[4]' (decision support facilities utilising Electronic Information Exchange System based on X.400 data structure) were undertaking substantial work in the area of Computer Supported Co-operative Work (CSCW) (Wilson, 91).

---

[2] Started in 1989 for ***CSCW research*** work in the European community.
[3] Organisations include TA-Triumph-Adler, Empirica, Jydsk Telefon, BNR Europe, X-Tel Services, Aarhus University, Rutherford Appleton Laboratory, Great Belt and GMD.
[4] Group Oriented Decision Support system at New Jersey Institute of Technology.

The USA National Science Foundation (NSF) is a leader in Co-ordination Theory and Collaboration Technology and is running several programmes in the field. The NSF projects include support for collaborative team (to develop a conceptual framework and a prototype system for collaboration in an asynchronous mode), and distributed group decision making (multi-disciplinary project for synchronous and asynchronous computer conferencing).

Other key projects are the Grace Conceptual Model for Group Communication (GRACE project described earlier) at Nottingham University, and at University College London, two projects: the XUA (JNT project) to design and prototype a work-station visual interface for X.400 (88) utilising X windows, and extension of PP (which is a X.400 implementation) for use of directory services, support of multi-media messaging, provision of secure messaging and implementation of message store.

A CCITT study group is also actively working in the area of Group Communication. The CCITT (study SG[5] VII or SG I Rapporteurs) Groups give the final shape to the recommendation.

2.4 Work Underlying The 'SAGE' Project

The literature shows that a set of basic requirements have been identified for a framework for a group communication base model. The 'SAGE' project uses the X.400 services and the definition of a basic group communication information model defined in the draft X.gc recommendations (Joint ISO/IEC/CCITT, 91) along with other important ideas discussed in various papers and projects. The research adopts the same definition of terms as in the X.gc working document, and ideas available in the literature. For example definition of item, entity, domain, role and object are the same as defined in

[5] SG VII & SG I are X & F series study group working on CCITT draft recommendations before approval.

CCITT draft document. The roles of moderator, group communication service agent and modelling of communication ports (discussed latter) are as described in the document. Terms which are not defined in the CCITT draft documents are defined elsewhere and if incorporated in the research are described in detailed.

The research in the 'SAGE' project builds on current knowledge to identify the services and tools for a set of group activities in an editing environment. The services would be considered as advanced group communication services in the context of its services and tools.

The concept of dividing the group communication system (a base model and a set of activities) (Benford & Palme, 93) is used in this research. The classification of group activities proposed by the 'SAGE' project takes a similar approach to Benford & Palme to define services and tools for a set of activities.

The paper by Benford & Palme (Benford & Palme, 93) describes persistency and structure of information as current issues. The research seek a solution to issues of persistency and structure of the information (structure of multi-version contributions). The 'SAGE' project research proposes three application group identifiers to recognise the services for each application group. These identifiers are proposed for three base classes (**Informative**, **Objective** and **Supportive**) as OSI applications.

The case studies are based on similar approaches in the AMIGO and GRACE projects and X.gc. The Newsletter activity consists of components defined in X.gc, AMIGO and other work. The components are structured in a similar manner, but some are modified and given suitable names. For example the 'moderator' (X.gc) is termed as 'group editing organiser' and similarly 'archive agent' (AMIGO report) is termed as 'supportive application group agent' with extended functionality (discussed later).

Chapter 3

X.400: Message Handling System (Overview)

3.0 Introduction

The 'SAGE' project research is built around the X.400 recommendations, so it is appropriate to describe the message handling system (MHS) services of X.400. Particularly, the research work is related to one of the recommendations (X.420) within X.400 series. The services recommended for group editing activities will assume the availability of a system with an X.400 implementation. For this reason it was necessary to closely investigate the X.400 recommendations and describe an overview of message handling system.

The model derived from the case studies adopts the features of X.400 recommendations, specifically the extension field facilities provided in the X.420 recommendation and the prototype used in the research is built around the features of the message handling system. The project is finally mapped onto the message handling system to produce the final outcome of this research: proposals for modification and extension in the X.420 recommendations.

Early systems for message handling were in the form of electronic mail. Electronic Mail has been available for around two decades (Tanenbaum, 89). The first e-mail systems simply consisted of file transfer protocols, with the convention that the first line of each message (i.e. file) contained the recipient's address. As time went on, the limitations of this approach became more obvious: it was inconvenient to send a message to a group of people, messages had no internal structure, making computer processing difficult, the

originator never knew if a message arrived or not, it was not easy to forward the messages, there was a poor user interface and it was not possible to create and send messages containing a mixture of text, drawings, facsimile, and voice.

As experience was gained, new proposals were made for more ambitious e-mail systems (Tanenbaum, 89). To prevent world wide chaos, the International Telegraph and Telephone Consultative Committee (CCITT), defined a series of protocols for what it called the Message Handling System (MHS) in its X.400 series of recommendations 1984. In 1988, CCITT modified X.400 to make it compatible with the Message Oriented Text Interchange System (MOTIS). MOTIS/X.400 is becoming widespread as the standard for all electronic mail systems. Both MOTIS and X.400 are concerned with all aspects of the electronic mail system.

The aim of MHS standards is to provide an international service for the exchange of electronic messages. Since 1984, the organisations CCITT and International Standard Organisation (ISO) which are involved in the production of standards for MHS, have agreed to harmonize their activities, and produce parallel standards with identical text. The CCITT uses the term 'recommendations' rather then 'standards'.

X.400 recommendations can be viewed both tactically and strategically. In the tactical view, the standard is seen as a gateway specification, a technological basis for connecting one message handling system to another. In the strategic view, on the other hand, X.400 is also seen as an architecture for the message handling systems and does not address the demands of the user interface.

The description of X.400 recommendations in this chapter will commence with functional overview of the X.400. This will be followed by description of the services (and protocols) provided in the application layer of the OSI reference model. Services

such as Interpersonal Messaging (IPM) X.420 and Electronic Data Interchange Messaging (EDIM) are also discussed as extended services of X.400. The last two sections describe the implementation of X.400 and its relationship with X.500 (the directory services).

3.1 The X.400 Functional Model

The X.400 message handling system (MHS) provides person-to-person communication on single user basis together with the distribution list facilities. Figure 3.1 shows the message handling system in the context of the layers above and below it. The MHS supports one-to-one and one-to many communication, but it does not support group communication between the users.

The 'SAGE' project describes a layer between the message handling system and the users to provide group communication support for a group performing an editing task. For example when five people are writing a paper then the system should set an environment for the communication (many-to-many) between them along with other special support to help complete the task.

There are two version of the X.400 series of recommendations. The X.400 (88) series of recommendations is an improved version released jointly by CCITT (International Telegraphic and Telephonic Consultative Committee) and ISO (International Standard Organization) in 1988 after the four years study period in which they harmonised their activities for the production of standards. The improved version is very similar to the 84 recommendations but is extended in terms of functionality. For example, it defines new entities like message store and access unit and also provides a mechanism for protocol extension to extend functionality (within IPMS) of the system. The description and, in general, the specifications of X.400 have become more formalized in the later (88) version.

The X.400 series of recommendations represents the Message Handling System (MHS) as an object which contains other objects. The objects contained are for example message transfer agent and user agents. The message handling system exists in a message handling environment. The message handling environment is shown in figure 3.1 which describes the relations between the components of the Message Handling System (MHS).

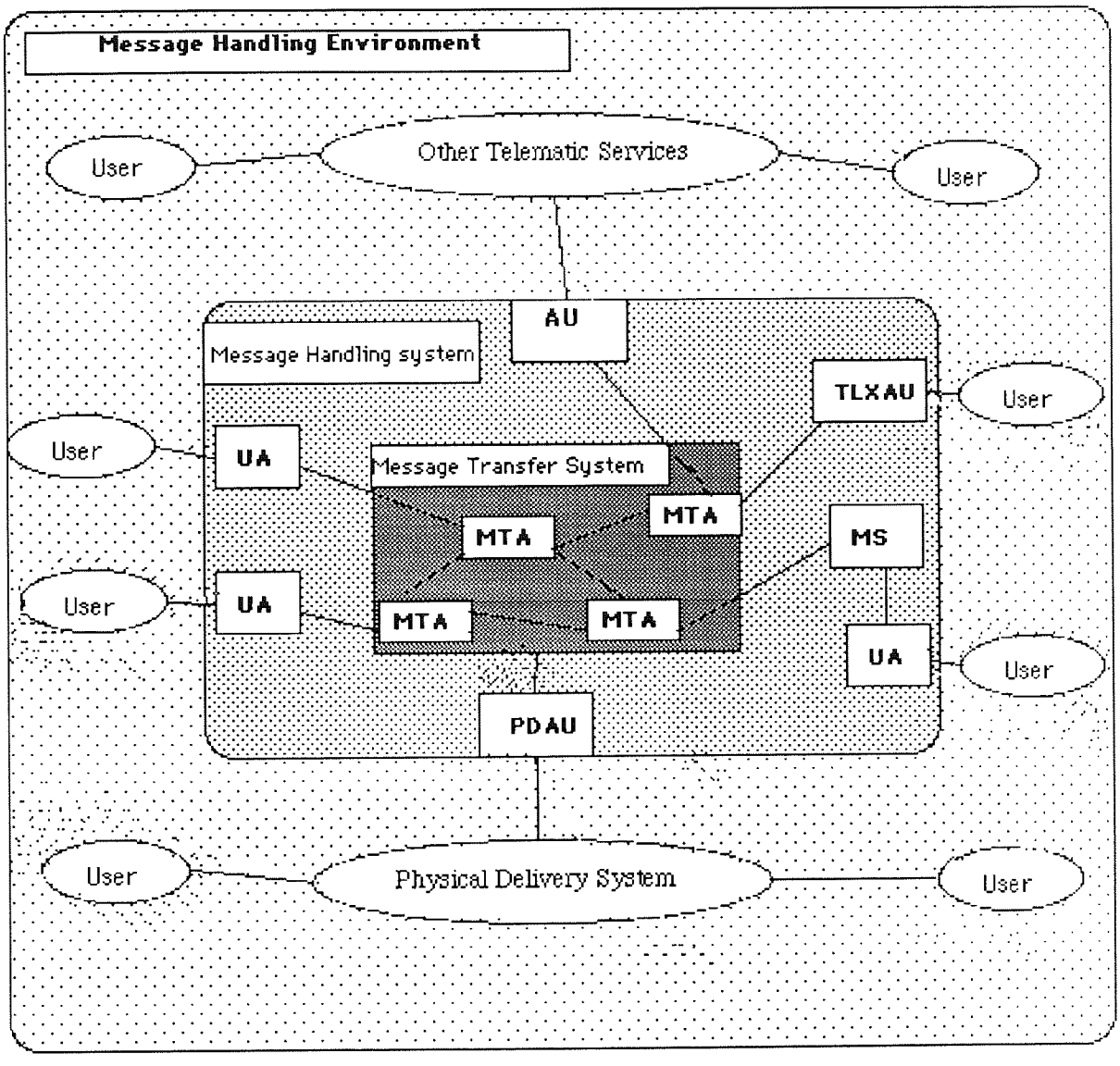


Figure 3.1 The Model of the Message Handling System.

The MHS is a layered model and the innermost layer, the message transfer system, relays messages from MTA (message transfer agent) to MTA. Messages enter this (innermost layer) transfer system via a submission protocol and leave it via a delivery protocol (Schicker, 1989). The middle layer, the MHS, contains the user agents that assist the message handling system user on the one hand, and engage in submission and delivery action with the message transfer agents on the other hand. The outermost layer, the message handling environment, is populated by the users who utilise the message handling system for communication of messages.

The message transfer system (MTS) is populated by Message Transfer Agents (MTAs). The MTAs accept service requests from user agent or from another MTA. There are two service requests taken care of by the MTS: a submission request made by a client, or a message received by transfer from a remote MTA. An MTA in receipt of a message from either source undertakes responsibility on behalf of the whole MTS for correct handling of the message. The inner part of the Figure 3.1 shows the functional view of the message transfer system. The block labelled MTA is a functional entity which, in cooperation with similar entities, conveys messages through the message transfer system (MTS).

The MHS layer contains User Agents (UAs), Message Stores (MSs), Message Transfer Agents (MTAs) and Access Units (AUs) (e.g. telex and physical delivery units). The User Agent (i.e. a functional entity) is an application process that interacts with the MTS or a message store (MS), to submit a message or accept delivery on behalf of a single user.

The Message Store (MS) is a functional entity whose primary purpose is to store and permit retrieval of delivery messages. The MS is an **optional** general purpose capability of MHS that can act as an intermediary between a user agent and the MTA.

The MS supports message retrieval and indirect message submission in the MHS. The message store uses the (message administration) services provided by the MTS. Each MS is associated with one user agent, but not every user agent has an associated message store.

The Access Unit (AU) provides a port to another communication system, such as a postal system. The Access Unit (AU) and, as a special case of the Physical Delivery Access Unit (PDAU) are indirect users and new objects in the enhanced version X.400 (88). They represent gateways to other services like telex, teletex and letter post. It is also conceivable that AUs may be used as gateways to non-X.400 systems (e.g. academic research networks such as the Internet or manufacturer-specific systems). The physical delivery access unit establishes a connection with the regular e-mail services.

The outermost layer is the message handling environment, and comprises the message handling system, user entities and the physical delivery services. The environment provides ways and means for multi-media exchange of information (e.g. text, teletex, fax and videotex). A process can also act as a user in the environment.

The X.400 (88) recommendation specifies the services of MTS in an abstract form (CCITT, 88). The message transfer system offers the other objects of the MHS abstract services for message submission, delivery and administration. The other view of the MTS considers all objects of the MHS as consumers[1] (Manros, 89) that make use of a service and consider the MTS to have the role of supplier.

A principle feature of MHS is that it operates in a store-and-forward manner. This means that the originator of a message need not wait until the recipient (s) indicates willingness to accept delivery before the originator sends the message. The structure of

[1] users are the consumers.

a message consist of two parts: Envelope and Content. The envelope contains all the information required by the MTS to convey the message to its intended recipient (s). The content consists of the substance of the message itself, supplied by the message originator.

3.2 Structure of the MHS Application

The X.400 (88) recommendations are aligned with the OSI standards. In analogy to the layering principle of the OSI-model, the message handling service is defined in two layers which co-operate via a protocol (Plattner et.al., 91). An abstract element of the layer which co-operate via a protocol are called entities. The entity which represents the functions of the User Agent Layer (UAL) in a User Agent (UA) is termed the user agent entity (UAE). Similarly, the term message transfer agent entity (MTAE) denotes an entity in an MTA which carries out the functions of the message transfer layer (MTL). These protocols and entities reside in the application layer of the OSI model. The services of message transfer system (MTS) and corresponding protocol building block are described in the form of application service elements (ASEs).

3.2.1 Alignment with Application Layer of OSI Model

The application layer consists of application services elements: Association Control Service Element, Remote Operation Service Element and Reliable Transfer Service Element specific to the layer, and five MHS specific application services elements (Message Administration Service Element, Message Delivery Service Element, Message Retrieval Service Element, Message Submission Service Element and Message Transfer Services Element). Figure 3.2 shows the structure of the application layer, as it is used for an application entity in X.400. The MHE-specific (message handling environment - specific) ASEs functions are described below. Figure 3.1 provides the MHE-specific services whose relationship with application services are shown in figure 3.2.

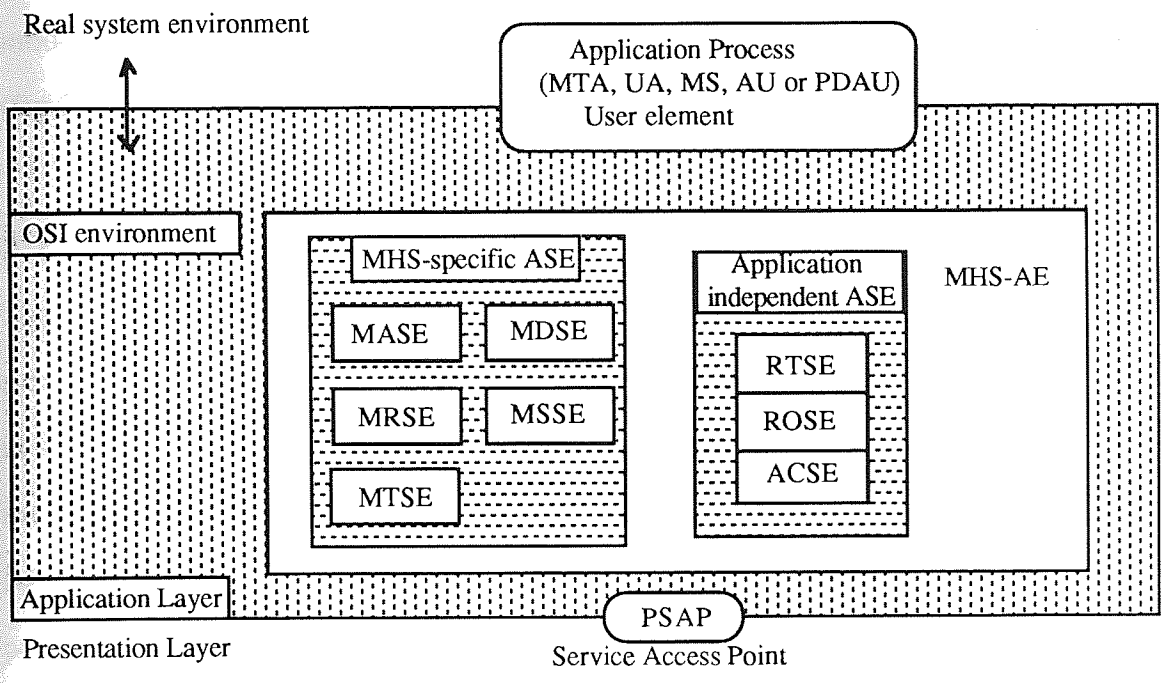


Figure 3.2 Structure of Application Layer for MHS.

The Message Administration Service Element (MASE) permits the registration of consumers (UA,MS and AU) with a supplier (MTA and MS) and the exchange of data for authentication. The Message Delivery Service Element (MDSE) delivers messages and reports and controls the delivery process. The Message Retrieval Service Element (MRSE) permits access to the message store through operations such as summarise, list, fetch, delete, register and alert. The last is used by the MS to signal the arrival of a new message. The Message Submission Service Element (MSSE) contains operations for the submission of messages and reports and for control of the submission process. The Message Transfer Services Element (MTSE) contains operations to establish associations and to transmit messages, probe messages and reports between MTAs.

The objects, User Agent (UA), Message Store (MS), Access Unit (AU), Physical Delivery Access Unit (PDAU) and Message Transfer Agent (MTA) in the functional

model of X.400 (88) represent the application processes in the sense of the OSI model (fig 3.1).

3.2.1.1 Protocol Used in MHS

The X.419 recommendation (X.400 series) defines communication protocols within the message handling system. The "P7" protocol is the access protocol between user agent and message store. The communication between MTS and MS or UA takes place through the "P3" protocol. The "P1" protocol controls the communication between MTA to MTA within the message transfer system. A detailed description of these protocols is given in Appendix 2.

Apart from "P1" and "P3" protocols the protocol "Pc" represents a class of application-specific protocols of the user agent layer (UAL) for the exchange of messages between user agent entities (UAEs). X.400 defines a single "Pc", the interpersonal messaging protocol "P2", which is in the context of interpersonal messaging system (IPMS) discussed in section 3.3. "Pt" is the protocol between UAEs and the user, in other words the description of the user surface that is the interactive terminal-to-system protocol (Plattner et.al, 91).

3.2.2 Features and Services of the MHS

Various mappings of functional elements onto real systems are possible, and user agents are available as application processes on a computer which itself implements message transfer agent functions. This is frequently the case for multi-user systems, which should offer a large number of UAs, and do this in the form of a user-callable command. The message transfer agent may also have a stand-alone implementation, as may the user agent.

The management of a world-wide MHS is a task which can only be successfully accomplished if the large system is broken up into smaller parts, for example domain part and local part (described in detail later). To this end, X.400 provides so-called **Management Domains** (MD). The X.400 series of recommendations defines two distinct management domains; Public or Administration Management Domain (ADMD) and Private Management Domain (PDMD). The recommendations cater for the standardization of the message traffic between administration management domains (within a country or across country boundaries) as well as the message traffic between public and private management domains. The address capabilities on the 'envelope' of a message allow for naming only one private management domain and this must be connected directly to an administration management domain. Any user agent that is connected to a private management domain has no direct connection to an administration management domain (Schicker, 1989).

In communication systems a **name** is a unique designation for an object. It is primarily its user who must be named. Similarly an **address** is also identified as an object but within the coordinates of the system. This address denotes the location at which the object may be found.

The message handling service is extended by connecting it to **physical delivery systems** such as the postal service. This allows for hard-copy delivery of messages originated within the message handling system, and in some cases allows for the return of notifications from the physical delivery services to the message originator. Physical delivery is an option which is also defined for the interpersonal message service. An example of this is the printing of a message and its automatic enclosure in a paper envelope. The access unit passes the physically rendered message to a physical delivery system for further relaying and eventual physical delivery.

The X.400(88) recommendations provide a basic form of group communication based on distribution lists (DL). Distribution lists are well-defined lists of subscribers (the members of the list). Messages sent to a distribution list are passed to all members. Distribution lists allow authorized subscribers to reach all group members of a given distribution list using a single address. The procedure by which messages are passed to the members is called expansion.

The message handling system offers various supplementary services over and above regular e-mail services. A subset of the supplementary services that transfer system and interpersonal messaging system can render is listed in Appendix 2. A complete list of all services is given in the annex B to the Recommendation X.400 (1988). The message handling system provides effective and better services: for example compared to a regular e-mail service, it is a simple task for an electronic message handling system to make multiple copies of a message by using the supplementary service "Multi-Destination Delivery" which allows the submission of a message with several addresses on the envelope. Similarly, the security supplementary service in the message handling system refers to the security of a message against casual or deliberate inspection by third parties.

3.3 Interpersonal Messaging System (IPMS)

Within the X.400 series, the X.420 recommendation defines the interpersonal messaging system. The purpose of discussing the X.420 recommendation in detail is that the research is built particularly on the X.420 features. The group editing services are mapped onto X.420 in a message handling environment as extended services.

As mentioned earlier a message is composed of an envelope and a content (shown in figure 3.3). The envelope part contains originator and recipients addresses together with all the information necessary to influence the transfer of the message.

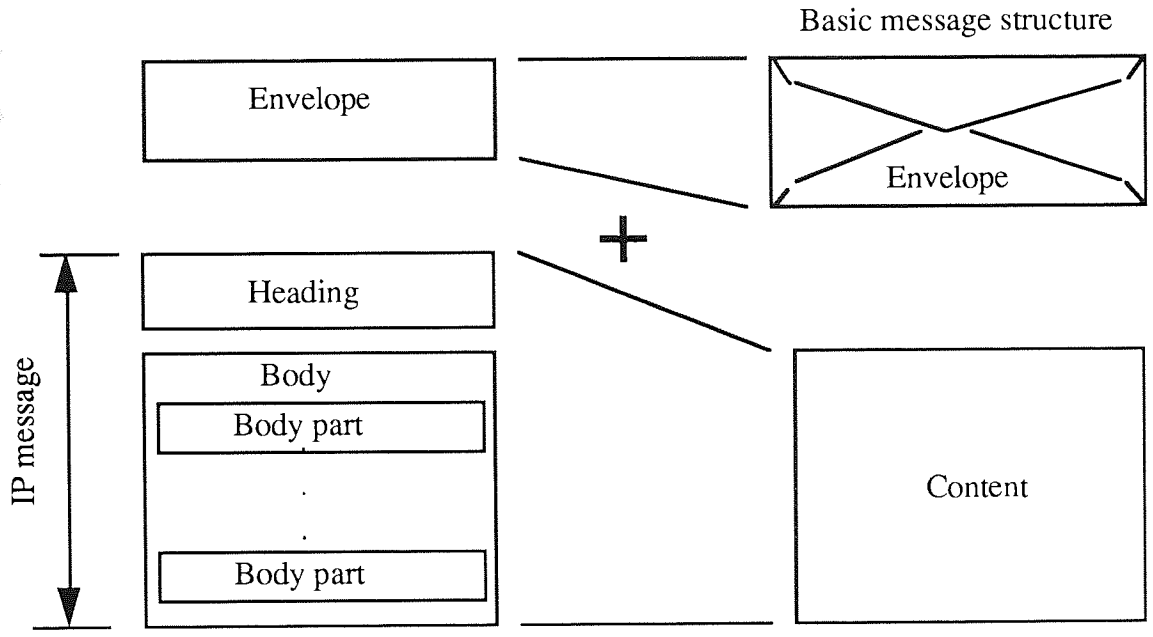


Figure 3.3 Structure of IP message.

The information is necessary to invoke the supplementary services of the message transfer system (e.g. request for a delivery notification). The X.420 recommendation is one of a set of recommendations for message handling. This Recommendation defines the message handling application called InterPersonal Messaging, specifying in the process the message content type and associated procedures known as "P2". The P2 content type is designed to fulfil the requirement of person-to-person communication. User agents are grouped into classes of cooperating user agents according to a common ability to handle messages of a particular content type. P2 determines the encoding of message content exchanged end-to-end by IPM-UAs and defines the semantics of information conveyed. The set of user agents in the IPMS constitutes the interpersonal messaging system.

The message transfer system (MTS) conveys user message from originator to recipients without regard to message content type. This simple model actually embodies two layers

of service. The message transfer service operates as a general purpose carrier of messages across the message transfer system. The P2 content divides into two parts, heading and body. The heading of an interpersonal message consists of a set of fields containing conventional items of information such as 'from', 'to' and 'subject'. The body of the message contains the primary information object which the originator wishes to convey. The message may consist of one or more body parts, each of which may contain different types of encoded information.

One of the important features of the X.420 recommendation is that it provides an extension mechanism for arranging new fields in the heading to extend the functionality of the protocols. This mechanism is based on the fact that the specification of the data formats at various places (for example, in a message envelope, in O/R addresses and in the heading of an IPMS communication) allows for additional fields in which protocol extension could be implemented. The protocol extension may consist of an arbitrary type and a value of this type. An example of the specification for an extension field in a message envelope is given in Appendix 2.

The X.420 recommendation defines the message handling application called interpersonal messaging, specifying in the process the message content type and associated procedures known as "P2". The recommendation within the message handling architecture identifies other documents in it. The recommendation also has a mechanism to extend the protocol and functionality. The purpose of describing the X.420 recommendation is to identify the group communication activities in the message handling architecture as other documents by extension and/or modification in the recommendation.

3.3.1 Services Obtained by Modified IPMS

Electronic Data Interchange (EDI) is an application using the X.400 service as a user agent service, much like that defined for the InterPersonal Messaging (P2) service. CCITT Study Group VII has defined a new content type and protocol for EDI (EDI for Finance Administration Commerce and Transport), currently known as Pedi (protocol for electronic data interchange) and reflected in CCITT draft Recommendation X.edi1 and X.edi2. The importance of EDIFACT here is that it is considered to be an application over X.400 to recognise another document like IPMS application.

EDI (Electronic Document Interchange) is the electronic interchange of documents in all possible areas of business. The aim of EDI is to replace the numerous paper documents used in the course of business by electronic forms. Such documents include orders, delivery notes, consignment notes, customs declarations, receipts and invoices. However, to ensure the interoperability of EDI systems from different manufacturers, international standardization of the electronic representation of all possible document types is under way. The standardization is collectively denoted by EDIFACT (Electronic Document Interchange for Finance, Administration Commerce and Transport) by the United Nations Economic Commission for Europe, ISO and other standardization committees. Detail information on EDI including file structure is given in Appendix 2.

The possibilities of identifying group editing services like IPMS and EDI as a document for a group of people are discussed later while mapping the services onto message handling system. However, The problem specification in this regard has been discussed in section 1.4. It is likely that the objects of the 'SAGE' project would be met with the help of X.420 recommendations.

3.4 X.400 Implementation as PP

Implementations of the X.400 protocols are now available widely and the implementation is known as 'PP'. PP is a message transfer agent oriented towards support of the X.400 series recommendation. The goal of PP is described in Appendix 2 which includes (Kille, 91): interface provision for message submission and delivery, support of all message transfer service elements and multi-media messages, support of multiple address format and use of directory services.

The ISODE (ISO development environment) is a key component of the PP implementation (Kille, 91). ISODE is an implementation of the OSI upper layers. It contains the layer services of session and presentation. These are common services: the ASCE, ROSE and RTSE, application services (described in Appendix 2); the FTAM and VTP and ASN.1 handling tools. The ISODE also operates over a TCP/IP network. This has enabled OSI applications, including PP and QUIPU[2] to be deployed ahead of provision of a global OSI network service. The ISODE creates an implementation environment for PP. ISODE is widely available, it is in public domain, which puts PP effectively in the public domain. The ISODE package (Kille, 91) has made the mapping of the PP message transfer service onto OSI a relatively mechanical operation.

A brief description on PP is given in Appendix 2 which includes an overview of PP, channel requirement and PP tables.

3.5 The X.500 (Directory Services)

In 1988 the standardisation bodies ISO and CCITT released the first series of recommendations X.500 for distributed Directory Services. The purpose of the Directory as X.500 standard is to supply a global name server and an application-

[2]QUIPU is the X.500 (directory services) implementation.

independent information service (Benford, 89). The Directories is a collection of open systems which co-operate to hold a logical database of information about a set of objects in the real world. The users of the directory, including people and computer programs, can read or modify the information, or part of it, subject to having permission to do so. The directory service facilitates the interconnection of information processing systems.The information held by the directory is, collectively known as the directory information base (DIB). Directories play a significant role in open system interconnection, whose aim is to allow, with a minimum of technical agreement beyond interconnection standards, interconnection of information system from different manufacturers under different managements and of different ages.

Directories are used to show the relationship of data elements to databases, files and programs. The Directory (some times also called Dictionary) is used to check all key automated systems for accuracy and duplication and permits an organisation to assess the impact of system changes on all automated resources (Black, 91).

A consequence of the growing number of message handling system (MHS) is that there is a need for a directory that supplies information about MHS users. In 1988 a working group was established by the standardisation bodies ISO and CCITT to define a distributed directory system of OSI applications and users. This is now available as the CCITT X.500 or ISO 9595 directory standard.

A detailed information about X.500 specifications and features of the directory services are given in Appendix 2.

3.5.1 Relationship to X.400

Message Handling Systems are expected to be the first and most important users of directory services. Only X.400(88) provides the prerequisite that the directory service

should be usable directly or indirectly in that it re-defines the term 'O/R name'. An O/R name in X.400(84) denotes an address, for which we use the term 'O/R address'. In X.400(88), an O/R name may now contain either an O/R address or a directory name or both. Directory names in the X.500 sense are distinguished names. Based on the new definition of O/R names there are several ways in which an MHS may use the directory services (Plattner et. al., 91):

(i) User-friendly name - The originator and the recipient of a message may be denoted by user-friendly names. If an O/R name consists of a name alone, this must be supplemented with the corresponding address using the directory service.

(ii) Distribution List(DL) - A group of recipients may be combined on a DL. This has an O/R address and also a name, if it is managed by the directory service. If the name of a distribution list occurs in the recipient field of a message, this must be expanded with the corresponding O/R addresses by the first MTA and replaced by the O/R addresses of the DL members, by the MTA responsible for the DL (expansion point). Both actions are executed using the directory services.

(iii) Functionality of MHS Component - The services supported by a UA, an MS or an MTA may be stored in its directory entry. If details of the functionality of a component are requested directly via the directory service, no probe message need be sent and the load on the MTS is lightened.

(iv) Mutual Authentication - Before the communication itself, the MHS components mutually authenticate each other.

(v) Interactive Use - The X.400 user may consult the directory service directly to find recipients and their O/R addresses. If one has the name he/she may determine the corresponding O/R addresses.

Figure A 2.8 in Appendix 2 shows the functional model of the relationship between X.500 and X.400. The use of a directory system is a local concern of individual MHS components; therefore, it has no effect on the MHS protocols and does not force the supplier or manager of another component to integrate these services. The model does not require a global directory system. Every component that wishes to gain from a directory system must possess a local DUA, since there are no protocols between X.500 and X.400. In Figure A 2.8 (Appendix 2), the X.400 components that make use of the directory service are shown.

Chapter-4

Group Communication Architecture and Models

4.0 Introduction

Group communications may be described as groupware, team work or co-operative work. Group communication may support users who are in a distributed environment or who share an information system. The group communication combines understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services and techniques (Wilson, 91).

The use of group work has evolved from a drive to increase personal productivity, which could lead to significant improvements in business efficiency and cost-effectiveness. Group communications should enable group of people to take advantage of networked information technology. To assess the benefits we need to understand why people form groups. There are many reasons, but among the more important are (Olson, 1989);

i) **Decomposing a task** -- divides a large task amongst a group of people, this may for example be to meet deadlines and distribute decision making applications.

ii) **Gathering relevant expertise** -- handles diverse aspects of problems with diversity of skills and experience.

iii) **Pooling diverse viewpoints** -- incorporates a larger set of ideas, wisdom, and judgements (as compared to the limited perspectives or opinions of individuals).

iv) **Performance** -- generalised improvement of individual's performance.

4.1 Group Communication System Architecture

The AMIGO report describes the group communication architecture model as a component which includes a group communication system (containing roles, rules and functions) (Weiss & Bogen, 89). The communication architecture model consists of information objects (e.g. a contribution) and the communication entities (e.g. group user) which form an *Information Model*. The functionality and services provided by the group communication system are defined in terms of the operations (e.g. create, delete and submit) in order to provide the underlying functionality to support a particular kind of group communication process (Weiss & Bogen, 89). These services are followed by the related support services (e.g. directory and message handling services).

The group communication architectural model takes into account three types of constraints (Weiss & Bogen, 89): the need to support 'distributed' group communication, the need to support high level descriptions of communication activity, and the need to support the dynamic nature of group communication. Within distributed communication, the communication architecture must allow the description of group communication systems that support multiple co-operating entities in the distributed environment. The scale of distribution should include office environments as well as international working groups.

To allow large-scale distribution a global naming scheme is necessary to identify uniquely communication entities (users, machine and process) and information objects

(messages, documents and information articles). At the same time the architectural model should support distributed storage to reduce communication overhead in a very large distributed environment.

The AMIGO report (Weiss & Bogen, 89) proposes that the group communication system should support the specification of multi-activities (e.g. add new activity) and operate under standard rules. The architecture model should provide integration with the activity model including its components (roles, rules and functions). That is, a proper relationship should be provided between the objects within the activity. The entities should exist within the architectural model. In addition to handling of the information objects by specific operations (e.g. submit, deliver, store and retrieve) these entities must support the handling of the role, rule and function components within the activity definition. Such activities should be specified by procedure oriented description.

The dynamic behaviour of the group should be maintained. The dynamic characteristics include allowing a new member to join or leave a group, or allowing the attributes of the members, such as addresses, tasks, rights or obligations, to change over the course of time. The configuration of group communication system must be dynamic (Weiss at el, 89). For example a service agent should not be permanently bound to one group or one activity.

4.1.1 Group Communication Information Model

The information model comprises the group communication environment (i.e. entities, domains and information objects). The Information model represents both information and members of the groups as objects (Joint ISO/IEC/CCITT, 91). The object classes in the model are arranged into a class hierarchy. Each group communication activity may define its own specific object classes, derived from the standard set of base classes. The Information model X.gc (Joint ISO/IEC/CCITT, 91) identifies three base classes: group

communication item, group communication entities and group communication domain. A group communication item is a basic unit of information (e.g. a message). A group communication entity is an entity which takes part in a communication process. However, the group communication domain provides the mechanism to bind groups of entities to the relevant information structures.

A group communication system might include the following services (Weiss & Bogen, 89) which are described in AMIGO report :

i) Interpersonal messaging to communicate with other users.

ii) Registration with the group communication system for different group communication facilities.

iii) Communication support services to interact with groups of users (i.e. collect, create, distribute, modify and read etc.).

iv) Filing and retrieval of documents.

v) Directory, to know where and how to access remote communication elements, applications or users.

vi) Encryption as defined in X.400 (1988).

vii) Authentication to avoid unauthorized access (personal, private or organisational)

viii) Asynchronous (store and forward) as well as synchronous communication mechanism.

ix) Information object transfer between different applications or remote servers.

The AMIGO MHS$^+$ report (Smith, 89) recommends the X.400 MHS, X.500 DS, FTAM (File transfer Access and Management) and DFR (Document Filing and Retrieval) as group communication support services. This means the group communication services must be based on Open System Interconnection (OSI) services. The group communication system assumes that the user would utilise these services to communicate with a group via a port which should form a group communication

environment. The group communication environment in relation to these services is shown in figure 4.1. The group communication system (GCS) may use the services of message handling system (MHS), or directory services (DS) or archive services (AS) or management services (MGMS) or all of them to fulfil the requirement of a group communication user (GC user).

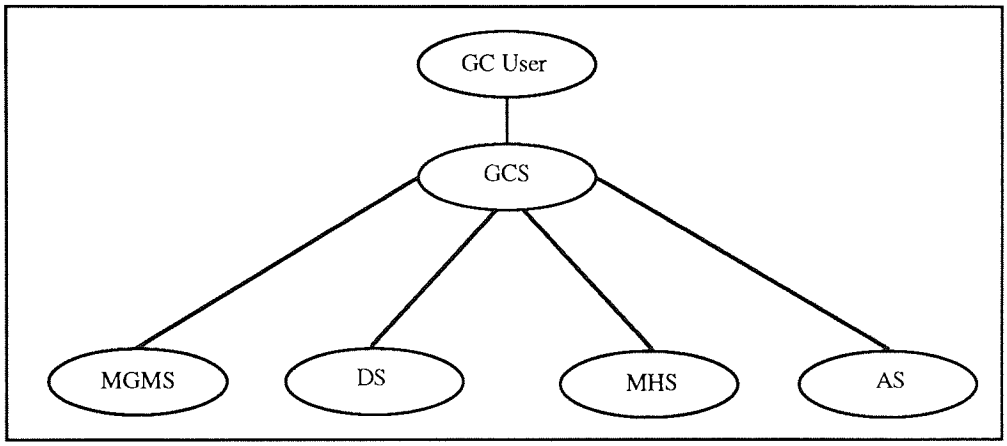


Figure 4.1 Group Communication Environment.

4.1.2 The Group Communication Services

The services offered by the group communication system may be categorised into basic services and advanced services: Basic services are realized within the group communication system, and may be viewed as common services. Advanced services are intended for handling some more detailed group activity procedures, for example: voting and group editing.

The group communication system basic service is made up of a **Distribution** service, an **Archive** service and a **Co-ordination** service, (shown in figure 4.2). Access to the services is made by special communication support system (discussed later). The functions of the three services (Weiss & Bogen, 89) are now discussed in turn.

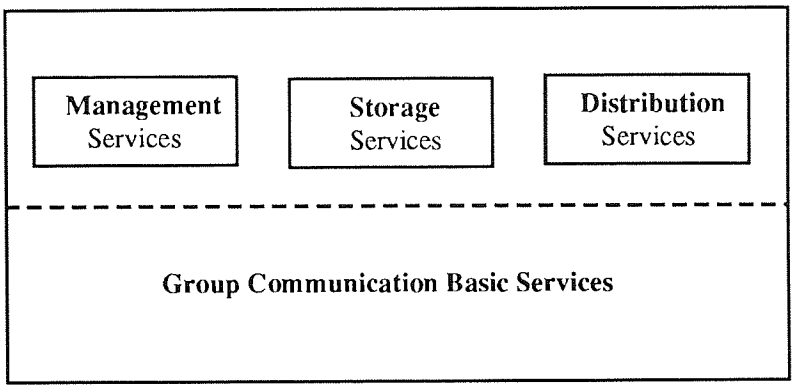


Figure 4.2 The Group Communication Basic Services.

The **Distribution** service provides the means of distributing group information objects (e.g. conversations, documents). A CCITT model of distributed group communication system is shown in Figure 4.3 which considers group communication system agents as the central functional entity.

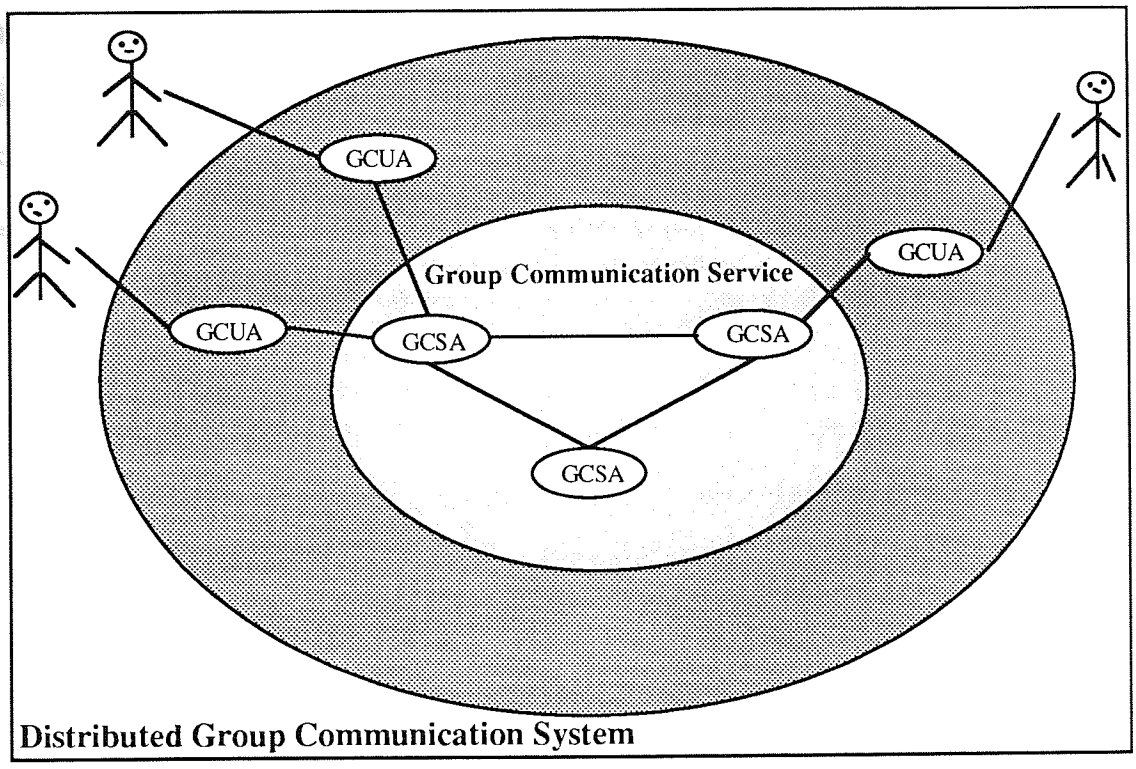


Figure 4.3 Group Communication System.

The service might be implemented in two ways: Firstly, by use of MHS Distribution List service (which may take the form of normal IP-messages (Interpersonal-Messages) being addressed to a group communication system agent (GCSA). In the architectural model a group communication user agent (GCUA) accesses the basic service ports via an interaction with a group communication system agent (Joint ISO/IEC/CCITT, 91).

Secondly, information might be distributed via the archive system ('pass-through', 'retrieve-to'). In order to use the distribution list service, a user access protocol is required. A system protocol specifying co-operation between distribution agent should also be defined.

The existing AMIGO report and CCITT draft provide no external **storage** services to fulfil the multi-user requirement. The design of multi-user store is addressed in the activities of ISO/IEC JTC 1/SC 18/WG 4 which deals with the specification of a document filing and retrieval (DFR) application. However, within CCITT's study group VII and the corresponding ISO standardization body, a message store (single-user) has been defined as an extension of the "P3" submission and delivery protocol (discussed earlier in section 3.2.1.1 of chapter 3).

Co-ordination service supports the co-ordination of the co-operating communication entities that provide group services. When a user interacts with the group communication system, the group communication system co-ordination service should interact with other services to achieve the goal. For example, distribution of a group information object might be achieved by the co-operation of entities such as directory service, a group information object transfer service (the MTS) and an archive service.

4.1.3 Group Communication

The CCITT draft X.gc (Joint ISO/IEC/CCITT, 91) definition of Group Communication "a service for computer-mediated human communication with special support for the interaction within groups of users". Group communication in a message handling environment may be realised as "co-ordinated interchange of information between sets of users connected via a store-and-forward message handling system". The information exchange activities require the provision of MHS services that handle Distribution Lists, Conferences, Bulletin Boards and more advanced uses such as Computer Supported Cooperative Work (Medina et.al., 1986). A group activity looks like a multi user system to the users who form a group to complete that activity in question.

Group communication has been defined by other researchers in different ways following the basic idea that it is 'the communication between more than two people' (Palme, 86 & 87). The CCITT view is that 'by group communication is meant a service for computer-mediated human communication with special support for the interaction within groups of users'. It may also be defined as the communication which fulfils the communication requirement for group of people working on a single pool within the boundaries of the group. The complexity of the group, may however depend on the group environment, for example forming a subgroup.

Group communication is carried out in three types of communicating environment: one user communicates with "n" users (one to many e.g. distribution list); "n" users communicate with one user (many to one e.g. distance learning); and "n" users communicate with "m" users (many to many e.g. group discussion). Where "n" and "m" are integers greater than one. This environment is created by a particular activity according to its functionality.

The group communication can be supported by developing a system in the context of distributed systems, distributed applications and system architectures (Klehn & Kuna, 91).

CCITT defined basic group communication services (discussed in section 4.1.1) to provide common support facilities to a variety of activities. In effect, the group communication system is a software layer which sits between different group applications and underlying technologies, providing services (co-ordination, management and a common storage facilities). The group communication system agent (GCSA) is considered to be a central functional entity which should represent the group communication system to the outside world. A group communication system agent is accessed via a group communication user agent (GCUA). The GCSA and GCUA capabilities and characteristics are discussed in AMIGO MHS+ (Weiss & Bogen, 89). Figure 4.4 shows an AMIGO functional model of the group communication system.

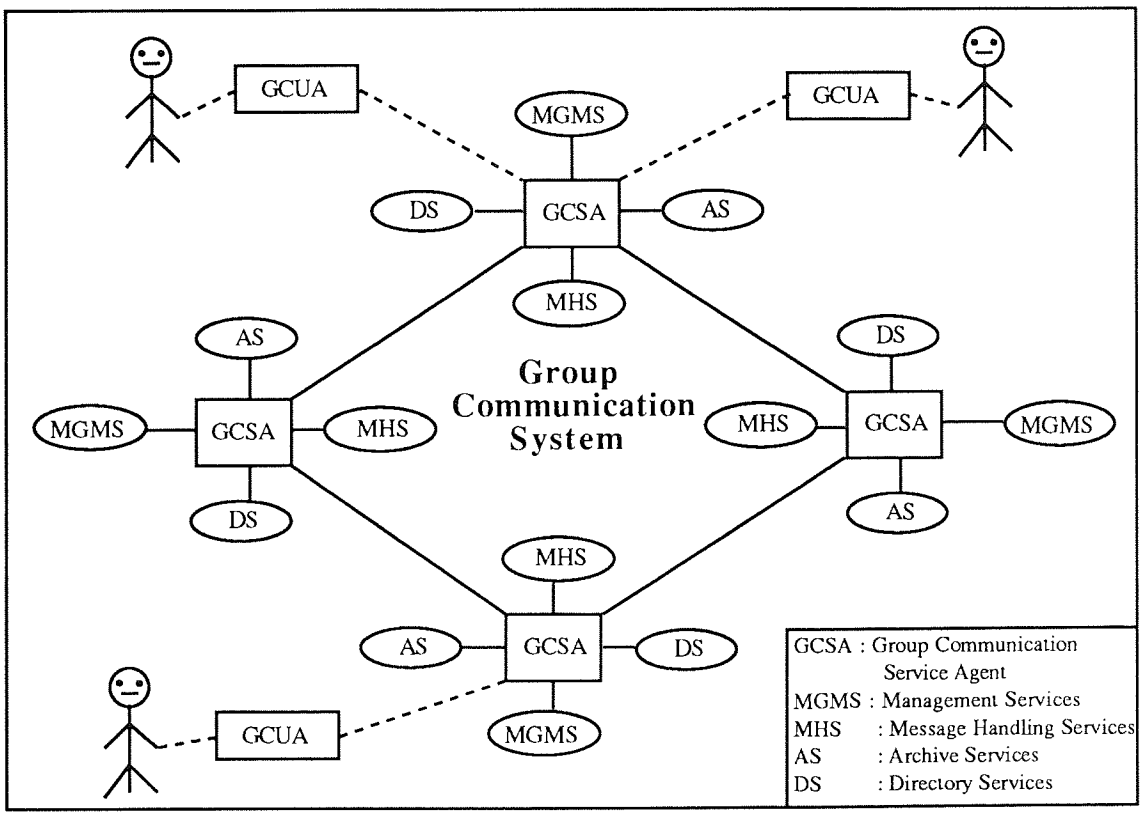


Figure 4.4 Functional Model of Group Communication System.

ASTON UNIVERSITY
LIBRARY AND

A complete list of capabilities and characteristics along-with GCS interworking with supporting systems are given in Appendix 3.

4.1.3.1 Group Activities

There are a large number of group activities. It may not be possible to provide services for each activity an individual basis. There may be a necessity to categorise these activities into various classes of group. We shall come back to the classification of these classes of group after section 4.3 and look into the possibilities for providing set of services tools for such classes.

4.2 Modelling of Group Communication

(Klehn & Kuna, 91) models the group communication into three sub-groups. A **cooperation model** describes the activity, in terms of cooperation, course of facts, the communication relations and the task distribution. This modelling is independent of the participating persons and of the environment where the cooperation takes place. An **organisational model** specifies the members of the group and its organisational structure, where a person can be assigned a role depending on their position in an organisation. An **environment model** determines the environment, where an instance of an activity takes place. It specifies the location of components and information relevant to the given instance of the activity.

(Jakobs, 91) introduces groups for different types of mechanisms to be provided by the group communication system. **Static group** represents the most simple class for example a distribution list, in which membership changes only rarely and which requires explicit management action to make the changes. **Semi Dynamic group** is not established explicitly by a list of members. The membership within this group is dynamic whereas the specifying criteria are static. The management costs and communication overhead are considerably higher because of changes in the

membership. **Dynamic group** comprises a set of participants having in common arbitrary criteria that were specified in a query. The membership and the specifying criteria are dynamic, which provides a higher degree of flexibility, but with very high cost and overhead.

4.3 The 'SAGE' View of Modelling

There are two main modelling approaches discussed above. The models (Klehn & Kuna, 91) by Klehn & Kuna are based on co-operation, organisation and environment. Jakobs (Jakobs, 91) introduces Static, Semi-Dynamic and Dynamic group classes for group communication. The classes or models are based on the nature of a group (viz co-operative group, organisational group static group and dynamic group etc.), but these group are not discussed in the light of group communication services and tools under any sub-group or class. Therefore, there is a need to describe the classes of group activities based on their functionalities to identify the services and tools for a set of class.

The 'SAGE' project takes a different view on classifying the groups. The project does not classify the group users. Instead, it classifies the group activities depending on their functionalities. This means a group has to be formed by the activities which are of similar nature: for example a set of activities which distribute information (bulletin board). These classes would then be able to provide the services and tools needed by each class in the group communication environment.

The 'SAGE' project introduces three classes of group communication models, for each of which there is a set of services and tools which meets the common functional requirement of the class. The classes are proposed as the 'Informative' application group, the 'Objective' application group and the 'Supportive' application group.

The first application group activity may fall in a group category which identifies those similar kind of functionalities which are of informative type: these activities distribute information and sometimes need replies. Examples of such activities are Bulletin Board, Distance Learning, Distribution list and multi-user System Broadcast. Typical activities of this category are listed under the head of 'Informative' in the table 4.1.

The second group category i.e. objective application group identifies those activities which need some kind of group decision making procedures, such as voting and executive decisions. Examples are producing international standards or executive meetings which need to take decisions to achieve an objective. In case of a difference of opinion there needs to be some kind of voting or other means to reach a single result (e.g. 'yes' or 'no'). These activities are listed under the heading 'Objective' in the table 4.1.

The supportive application group covers cases where groups of people work together on a single pool to complete certain tasks for an activity. The activities are categorised under the **supportive application group,** which needs supporting work within the group and cannot be completed without help of the rest of the group members. Examples of such activities are cooperative writing, a software development team or producing minutes of a meeting. The activities under heading 'Supportive' in the table 4.1 identify the application group. Some of the functions of supportive application group will appear in other groups.

Group communication comprises a large number of group activities. Some activities are listed in table 4.1 under three different headings: Informative, Objective and Supportive. The list is compiled from AMIGO, GRACE and other literature.

Informative	Objective	Supportive
Bulletin board	Date Planning	Group editing
Distance learning	Conferencing (specific)	Distributed S/W updating
Lecturing	Project Managt. Co-ordination	Meeting environment
Presentations	Producing International Standard	Editing Newsletters
Course Assignment	Executive meetings	Questionnaire session
Procedures	Election	S/W design team
Help-Desk	Opinion Polls	Minutes of meeting
Seminars	Voting	Auction
Sales & Buying	Experts Opinion	Stock Market

Table 4.1 Classification of Group Activities.

The proposed three base-class group applications are based on the functionality of the group activities. The 'SAGE' project proposes three innovative classes of application groups. The application groups are proposed to be identified as three different applications in the OSI environment. As with the InterPersonal Messaging (IPM) service in the X.420 recommendation, there is a need to define (by extending the functionality) three application types within the heading field component type to recognise these application groups. The services which are to be met by these three application groups are proposed to be called InformativeGroup, ObjectiveGroup and SupportiveGroup Messaging services. These services would have their associated service agents as in X.gc (Joint ISO/IEC/CCITT, 91) to fulfil the requirement of each application group. The underlying functionality for these proposed application groups would be met by their respective application group agents. For example, the services (comprised of basic, advanced and information handling) required by supportive application group is to be met by the supportive application group agent (SAGA).

The position of Informative application group, Objective application group and Supportive application group within the application layer is shown in figure 4.5.

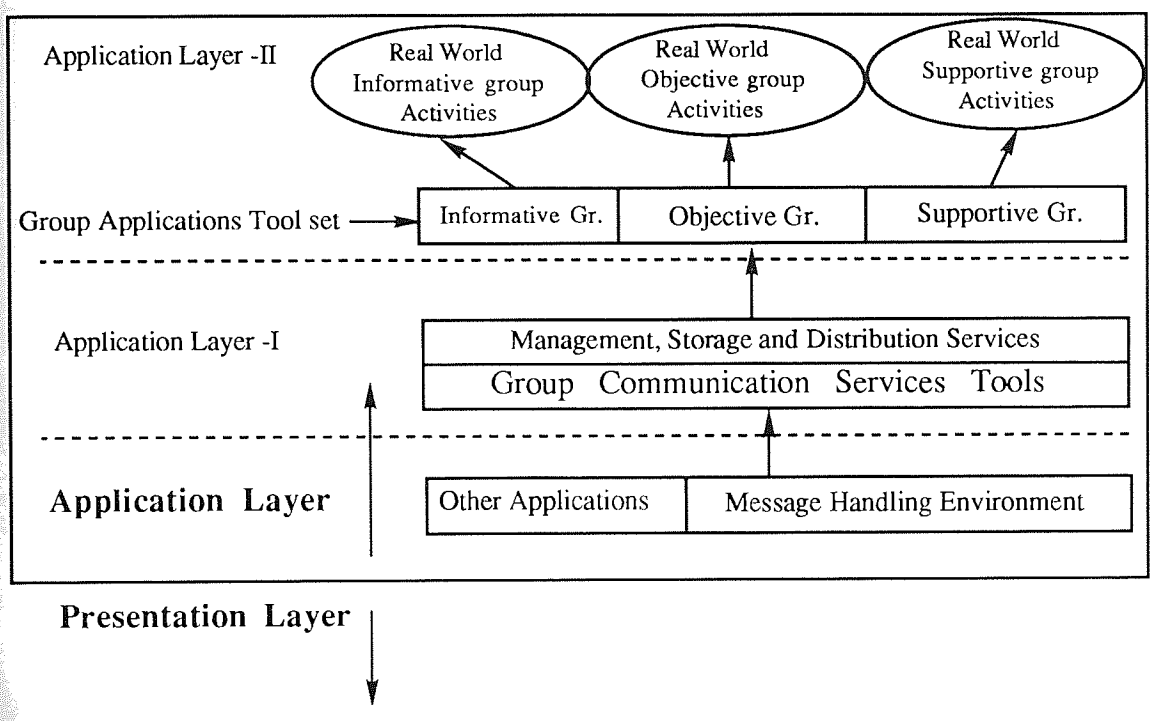


Figure 4.5 Layering Structure of Application Group Services & Tools.

Taking this view, a group communication environment should contain the information objects, the functional or operational objects and the entities. The relationship between these would form a group to complete an activity. Hence these objects and entities should not belong to any specific class (e.g. domain, activity or group). However, the underlying functionality for an operational object is dependent on the nature of the activity. For example, functionality of the 'add-member' operation is different in the Supportive application group and Informative application group. The functionality involved in an Informative application group different in its access support (read, write and delete) to the Supportive application group. The application service agent would be able to provide the required functionality for an operational object in each case.

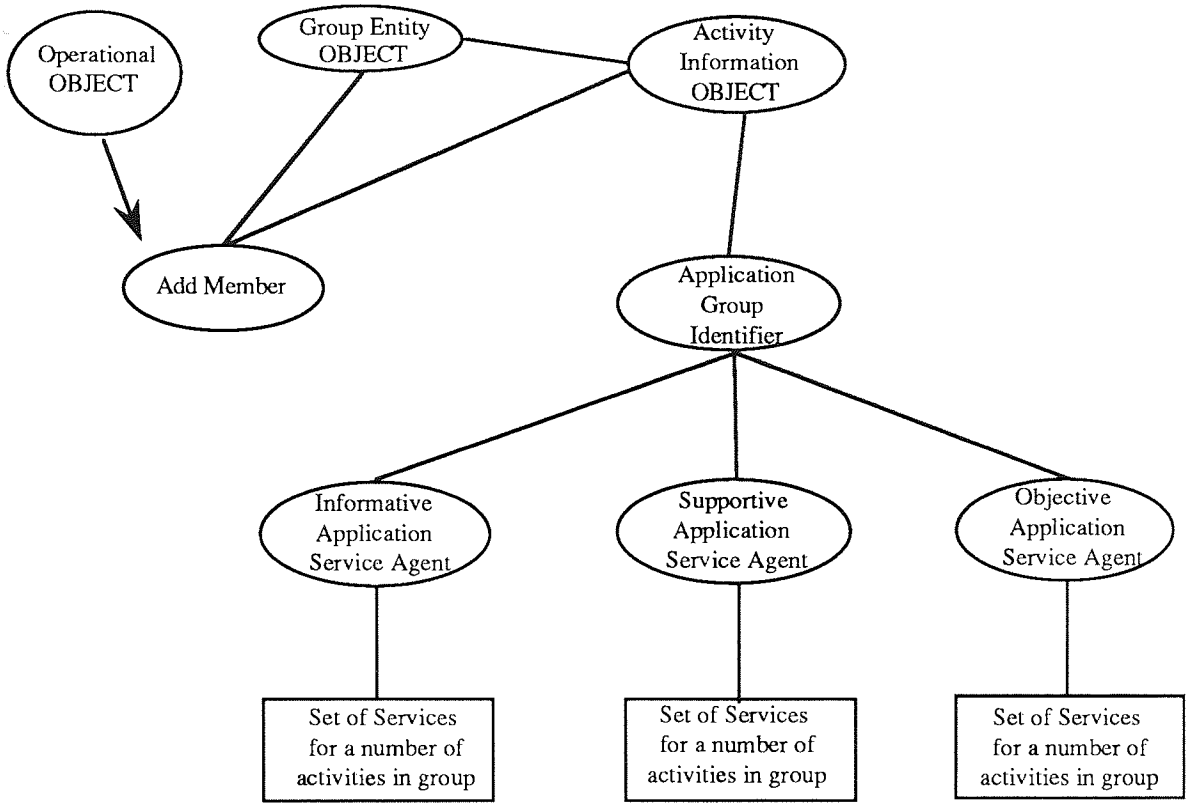


Figure 4.6 Functional Object Relationship with the Activity.

4.3.1 The Functional View of Operational Objects

Figure 4.6 shows the 'SAGE' project's relational view of an functional object with the activity and group (which is a set of the entities), in which no component is bound to any other one, and free to form any type of group communication structure. The following are the important implementation issues involve in the proposed configuration:

i) The functional objects should be able to identify the activity,

ii) The activities should identify the particular class of application group,

iii) The application group should select the required application group agent, and

iv) The selected group agent should provide the services and tools required in that class.

4.3.2 Modelling of All Application Groups

The goal of Group Communication Information Model X.gc (Joint ISO/IEC/CCITT, 91) is to provide a framework for implementing a variety of group communication applications. A requirement of a model must be able to represent many different types of objects: such as documents (e.g. message, newsletter and reports etc.), filing (e.g. notice boards, topics and conversations) and users (both individual and groups). The objects might have complex structures and relationship. The model must reflect the administrative boundaries and must allow the implementation in a distributed environment.

The group communication environment can be modelled as a functional object, functions of which include reading a contribution, editing a contribution and storing a contribution. It can be subdivided into three classes of group communication models defined earlier. Diagrammatically it is shown in figure 4.5, within application layer II as a group communication services and tool set under three different heads: Informative, Objective and Supportive application groups. These application groups should be able to meet the functional requirement of the activity which is identified under such application group. The functionality requirement (basic and advanced) includes operational and rule-based information handling requirement.

The purpose of the case studies to be discussed later is to identify common features between these activities. For example, the activities of editing a technical paper, or a editing Newsletter or preparing the minutes of meeting are likely to have common functional requirements. These requirements will include such things as version control, access control, storage management and security management. The tools which meet these common functional requirements should be designed in the context of the needs of the full range of group activities rather than in the context of only one.

This chapter describes the group communication architecture and its information model. The role of support services is also described in the light of group communication basic services. Different modelling views of group communication are compared in the chapter. Finally the 'SAGE' view of modelling of group activities is discussed in the light of group communication services and tools.

Chapter 5

Case Studies & Identification of General Model

5.0 Introduction

This chapter describes the purpose of case studies and the method used for the case studies. The method used for case studies involves user interaction to formulate the specifications for the prototype and to deduce common features between the activities. To investigate common functionality, activities are selected from a particular topic (i.e. editing) within the supportive application group (discussed in section 4.3, chapter 4). The problem definition for these activities, viz a Newsletter, a Technical paper and a Software development team, is discussed in this chapter.

This chapter also describes a general investigation and a theoretical solution of a conceptual model for group editing. The conceptual model is discussed in the light of issues such as: editing environment, role of group editing organiser, access control, version control and message flow mechanism.

5.1 Why Case Studies are Required

Support of group communication requires hardware development, programming development and information development, together with successful user tools and/or acceptable standards. The success of such services, tools and standards depend on user support and satisfaction. To develop a software system (including prototypes), one way to start is by studying the 'real world' way of working by looking at existing manually based systems. The study of manual systems reveals a variety of methods and operations needed to complete the activity. The end users of group communication

system services are people of the 'real world'. Therefore, it is felt necessary to obtain the views of users who are concerned with such services and tools.

The case studies aim to deduce underlying requirements by looking at how group activities are performed. This is achieved in two ways: by study of related literature and by user interaction. Study of the editing process in the literature establishes how group editing is described in terms of its working processes, and user interaction establishes how people achieve the final shape of the edited document.

The case study approach follows methods used in the development of X.400 standards from e-mail systems, which involved studying existing e-mail systems and interaction with those people who were working with e-mail systems, and on this basis determined useful recommendations.

As described in section 4.3 of chapter 4 the 'SAGE' project divides group activities into three classes of application groups: Informative, Objective and Supportive. The cases selected for study are within the supportive applications group. Specifically group editing activities are considered as a subset of the supportive applications group. The cases produce the results of a study of 'real world' cases in group editing. From these cases common characteristics and features editing can be deduced.

5.2 Method of Case Study

In seeking a systematic approach to software development, the software industry is recognizing the need for a variety of new practices. Structured methods are accepted as necessary if large systems are to be developed. Structured methods provide notations to aid and capture the functional specifications of the requirement. The notations also provide the notion of abstraction which is required for the application of a structured method.

The first step is to produce an abstract specification characterizing the essential properties of the problem and stating what is required. In producing an abstract specification, it is necessary to understand the problem structure exactly. The enquiry process (Checkland & Scholes, 90) is a soft system methodology which reflects current 'real world' practice. The enquiries are logic-based and cultural. The case study considered uses a process of modelling in a methodological cycle having the following stages (Checkland & Scholes, 90): 'finding out', expressing the problem situation, formulating root definition, building concept models, comparing models and the perceived 'real world', debating definition changes and taking action.

To form a conceptual model of the problem, there are many questions to be answered, for example: problem description, interaction, extra facilities, and integration. In the 'SAGE' project, the approach used to produce an abstract specification involved an initial 'finding out' phase which proceeded by personal discussions with the users of editing activities. This results of the investigation were enhanced with knowledge taken from the literature. 'Finding out' at an early stage in the development was found useful to produce a basic structural design of the problem situation and a detailed model of a general system. To express the problem situation three relevant (within supportive group) activities are considered for study: production of a Newsletter, production of a technical paper and software development by a team. In the phase of formulating the root definition, a general solution is derived which is considered to be a common for all such problems. The 'building conceptual models phase' describes the selected activity models of the chosen activities, and discusses one of the models in detail. Stage five and six of Checkland & Scholes approach are considered in reverse order: the detailed model is prototyped and demonstrated to the 'real world' and then these models and the outcome of the demonstration are evaluated to derive a final model for group editing activities.

5.2.1 Conduct of User Interaction for case studies

The following people were interviewed during the case studies: Dr. H. Shah (for general editing background), Dr. D. Newman (involved in CSCW activities), Dr. B. Gay (former editor of academic journal), Dr. P. Coxhead (book editor), Prof. D. Avison (author and journal editor), Dr. Celia O'Donovan (editor, departmental newsletter), Mr. G. Bellavia (for knowledge of software development) and Ms A. Grant (co-editor, Aston Fortnight).

Pre-planned interviews were arranged on an individual basis. The interview subjects were visited in their work places. A brief description of the project was given verbally, followed by discussion of editing issues in general. A discussion guide had been prepared before the interviews. The guide addressed general and specific issues on editing of a Newsletter, editing a technical paper and software development. More specifically the following issues were raised:

i) Facilities to avoid redundant editing;

ii) Version control;

iii) Version requirements;

iv) Access facilities;

v) Role of editor, co-editor and guest editor and control over document;

vi) Editing process;

vii) Collection of information and storing the information;

viii) Naming, creating, processing and completing a Newsletter process;

ix) Header information requirement;

x) Adding new members and their contributions;

xi) Issuing Newsletter and its distribution;

xii) Formation of software team;

xiii) Information flow;

xiv) Upgrading to new version; and

xv) Testing and integration of a package.

The issues were raised in the order listed, with some alteration for different activities. All were presented with the common issues (i) through (vi). Additional issues related to the Newsletter (vii through xi) were raised with people working with a Newsletter. The issues (xii) to (xv) were raised in the context of the software development activity.

These issues were put to the interviewee in the form of verbal questions. During the interview, cross questioning was done wherever answers were not clear or clarification of the issues were needed. Each interview lasted an hour or so. Responses were recorded manually.

A summary of the user responses now follows.

General issues (i) through (vi):

To avoid redundant editing, it was suggested that various editing mechanisms be adopted: page editing, screen editing, line editing, where page, screen or line are locked during editing, or locking the document itself. There were mixed responses about editing a contribution where there is more than one version: one set of responses favoured editing of any version of a contribution available on a topic, while another set favoured allowing editing only of the last version of the contribution. Most favoured having available two to three versions of a contribution on a particular topic.

In the present manual system, only the editors have access to a contribution. In a group editing system, it was thought desirable to have read only access for all members of the group. However, write access should be provided to the author of the contribution, while the editor should have all types of access.

The users felt that the Newsletter should centrally be controlled by an editor, operating with co-editors if needed. A co-editor may form a sub-group and should be able to handle the sub-group independently within the group.

Newsletter issues (vii) to (ix)

All information related to a Newsletter should be stored under the name of that Newsletter, so that it can be made available in the future. A distinguished name should be given to each Newsletter. For example, 'Aston Fortnight Vol.3 No.8', where Vol. represents the year and No. represents the number of issue in that year. A completed Newsletter should be closed and no information should be added to it. A header for a contribution should contain information about the Newsletter, for example, submission date, name of the Newsletter, topic etc. There were mixed responses on whether the status of the member (i.e. reader, author and editor) should be included in the header.

Users considered it should be possible to add a new member to the group at any time. A provision should be made allow the editor a convenient way to generate the Newsletter in any specific order from the available contributions.

There was some support for automatic distribution of the finished Newsletter to its readers. A bulletin board approach was also suggested.

Issues related to the software development: (xii) to (xv)

A team should consists of the members who are related to a package which bound by single specifications. The distribution of tasks to the team should be made on modular basis. There should be a provision for the flow of information between the members of the group. The information flow should include data dictionary, test data and global parameters. Only tested modules should be submitted to the editor. These modules may

be tested with simulated data or test data. Sufficient testing should be carried out as integrated package before distribution for implementation.

The user interviews were important in understandings the problem definition and underlying functionality. The interviews provided sufficient information to form specifications for the prototype which were enhanced with the help of the literature.

5.3 Selection of Group Activities for Case Study

The common features and characteristics for group editing issues were investigated by selecting the cases in one application group (i.e. supportive application group) with similar functionality (i.e. group editing). The selection of group activities for investigation has been made on the basis that the activities should have a requirement for advanced services and the activities should also have some commonality with each other (e.g. editing documents of different characteristics). Although the cases selected for study all fit under the heading of group editing, part of their procedural operations may be different.

The cases chosen for study are editing a newsletter, writing a technical paper and software development by a team. All cases are considered to involve a group writing in a distributed network environment. In the 'SAGE' project, the common characteristics of cases which would be useful for defining group communication services and tools are described later.

5.3.1 Editing a Newsletter

A Newsletter is a document which is produced jointly from the contributions on different topic/subjects by its authors and editors. The Newsletter has readers who are the beneficiaries from it. This is a group activity and is headed by a chief editor. The Newsletter editing activity in the 'SAGE' project is managed by a group editing

organiser, who is empowered to make changes and/or take decisions. It is not necessary that the group editing organiser agrees with the contributing author(s). Thus editing a Newsletter may be viewed as a centralised control activity. The Group Editing Environment may however consist of nested sub-groups in geographical or other domains, and operational powers may be distributed within the group or sub-groups.

5.3.2 Editing a Technical Paper

A technical paper written by a group of people is a joint editing activity. The object of the activity is to produce a final version of a paper which is written on a single subject/topic. Each member of the group may have equal status in the group. However, in comparison to a Newsletter, this activity may be viewed in different ways in terms of power and control. The group editing organiser in this case may have to take into account the views of author(s) of the paper. For example, after a referee's comments have been received the group editing organiser and the author(s) have to agree with each other before publication. In this application the role of editing organiser is weaker than the earlier one, and control is distributed among members of the group.

5.3.3 Distributed Software Editing Team

Software development by a team is also a group activity. The group activity is critical in ensuring proper testing and integration of a package in a distributed environment. The members of the activity also need a high level of interaction between them, and frequent reference to the specifications. This case may however be viewed in another way in comparison to the other two cases. Each member of the group has a specific task to edit, whose parameters are pre-defined to form an integrated package. The members have to take care with interfacing parameters of the package. The group editing organiser in this case may probably have to act as co-ordinator among the team members. The group editing organiser may or may not agree with the member concerned. In case of

disagreement the group editing organiser may have to work on the member's contribution. The role of GEO may be considered to be between the role in Editing Newsletters and the role in Editing a Technical Paper.

5.4 The Group Editing Problem

The 'SAGE' project seeks to recommend group editing services and tools to provide a process for group communication system. This requires identifying functions, operations and their relationship in a group communication environment (for example). The case studies deal with the various aspects of the handling of editing issues for a document in a group environment. These aspects are, for example, Updating, Co-ordination, Version control, Access facilities, and Document handling. The case studies are also used to identify the common functionality between editing activities. A working prototype is developed as a user testbed and to clarify detail of services identified during the investigation phase. The case studies and prototype provide vehicles for identifying needs and developing and evaluating proposals for support services and tools.

To decide what facilities the system should provide, there are several questions to be addressed. Aspects not directly related to this problem (in context of X.400) are not discussed here (for example; message composition tools, final page layout, advance planning, cross-referencing, (auto)indexing). Aspects which are directly related to the messaging system however, are considered in depth. These include message communication frequency, automatic message delivery, studying and modifying draft, subsequent reviewing, full text retrieval, new version distribution, number of versions, reminders facilities, editor/author relationship, access facilities, refereeing, version modification, notification and storage facilities.

The discussion of the case studies is divided into three parts. The first part describes a conceptual model in general terms for group editing activities. This part is described in

this chapter. The second part discusses the Newsletter case study from which the specification requirement for the prototype is derived. The last part looks at the case studies of a technical paper and software development team. The last two parts are discussed in the following chapters. The final outcome is derived from the solutions of all cases and the user comments received on the testbed prototype.

5.5. Conceptual Model for Group Editing

Group editing is a procedure that allows a group of authors/editors to produce a document in the form of master document in an acceptable form. The group communication information model (discussed in chapter 4, section 4.1.1) identifies three base classes (item, entity and domain). A group communication item is a basic unit of information which is exchanged or manipulated during a communication process. Typical items are: messages, notifications, suggestions, contributions, chapters and documents. A group communication entity is an entity which actively takes part in a communication process. Examples are group editing organiser, authors, readers, co-ordinators, editors and agents (a process can be a user) (Joint ISO/IEC/CCITT, 91). Group communication domains provide the mechanism to bind groups of entities to the relevant information structures for such editing activity. They may represent specific group communication activities. (e.g. topic in group editing). A domain may contain entities, items and other domains.

Figure 5.1 shows elements of the information model X.gc (Joint ISO/IEC/CCITT, 91) containing the components of the model. The information model consists of domain A & B and sub-domain within domain B. Each domain and a sub-domain consist of messages, conversations, items, agents and entities as described in the three base classes.

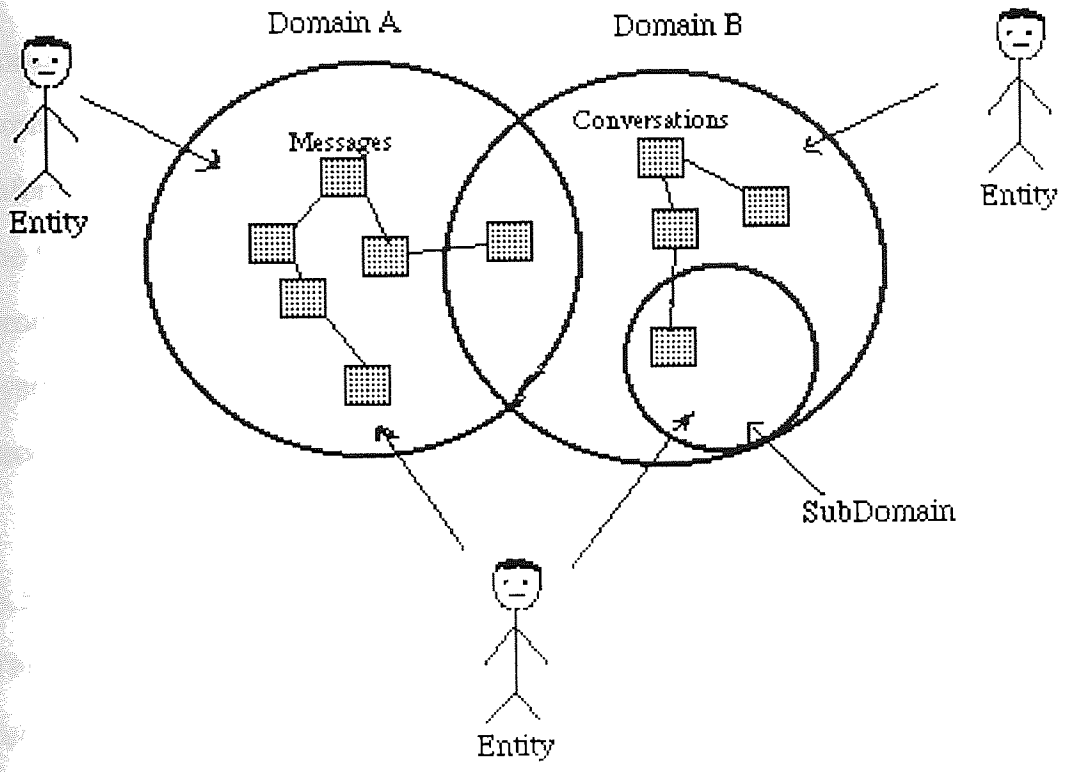


Figure 5.1 Elements of Group Editing Model.

Group editing should provide special support for a manuscript by several authors (Palme, 88). It may deal with joint storage area for the text and control of simultaneous updates. It may be based upon the tree structure of the manuscript where an update operation applies only to a certain node of the tree structure. The document is structured into a detailed hierarchical structure (Smith et.al., 89) where each message may be considered as a node. Each node may be handled in the same way as message. The message may refer to other document and may contain additional special field for group editing for example, 'part of', 'obsoletes' (Wagner & Palme, 89) and 'commented' etc. The existence of the group has to be maintained until the end of activity.

5.5.1 Editing Environment

An editing group is to be formed within Group Communication Environment. As described in X.gc (Joint ISO/IEC/CCITT, 91), a Group Communication Environment (GCE) can be modelled as a functional object. Since a group editing environment is

considered to be a sub-set of GCE, it can also be modelled as a functional object. The group editing environment is the relationship of the group members for manipulating information objects in an editing activity. The environment needs to define the roles and the functions or protocols adequate for accessing these information objects. Objects of group editing include a single central object (e.g. a contribution), the group editing system and the group members. A group member may be a person and/or a computer process, who can originate/receive group editing information.

5.5.2 Group Editing Organiser (GEO)

The group editing organiser is a person with special powers to operate the group activity. As described by Palme (Palme, 87 & 88) the operations may be to remove irrelevant messages, add/delete members, restrict access and perhaps control over contributions. In accordance with the CCITT draft release working document X.gc for group communications (Joint ISO/IEC/CCITT, 91), it is assumed that the group is 'controlled' by an entity called the 'Moderator'. A similar functions is carried out by 'Group Editing Organiser' (GEO) in the 'SAGE' project. The GEO entity is necessary for the smooth running of the group and to fix responsibility. The powers of Group Editing Organiser will be discussed further in later chapters.

The Draft X.gc (Joint ISO/IEC/CCITT, 91) describes three communication ports to meet the functionality requirement at three levels. Access to the services is made by the concept of a port in the communication system. A port may be defined as a point at which an abstract object interacts with another object, for example, the 'add member' service would interact with an activity object and a group object to create a relationship with the new member with whom it is to be added. The draft X.gc (Joint ISO/IEC/CCITT, 91) in asynchronous computer conferencing defines these three level as member, moderator and conference manager. The 'member' port provides the functionality associated with members (e.g read). The 'moderator' port provides

additional functionality for moderation (e.g. delete). The 'conference manager' port provides the additional functionality for administration (e.g. create).

The names (in terms of their roles) of the entities in the 'SAGE' project differ from those used for computer conferencing in the X.gc draft, reflecting slightly different functions. Figure 5.2 shows the mapping of CCITT entities with the editing group entities in the 'SAGE' project.

CCITT	SAGE
Conference Manager	Group Editing Organiser
Moderator	Coordinator/Editor
Member	Author/Reader.

Figure 5.2 Mapping of Editing Entities with CCITT Draft.

The mapping of group editing system communication ports viz member, co-ordinator/editor and group editing organiser is shown in figure 5.3. The group editing environment is considered as a group editing system where its functionality would be met by the service agent within the application group (discussed in chapter 4).

Considering the wider range of the group and making use of concept described in AMIGO report, it is assumed that the group contains sub-groups. For example there would be a sub-group of editors, and/or sub-group of a technical paper writing team. In such cases the group editing organiser may have a co-ordinator for each of sub-group. It is understood in the 'SAGE' project that the sub-group editor or co-ordinator would, within a sub-group, have powers similar to the group editing organiser. The group editing organiser should be able to assign such operational powers to intermediate entities (co-ordinators or co-editors).

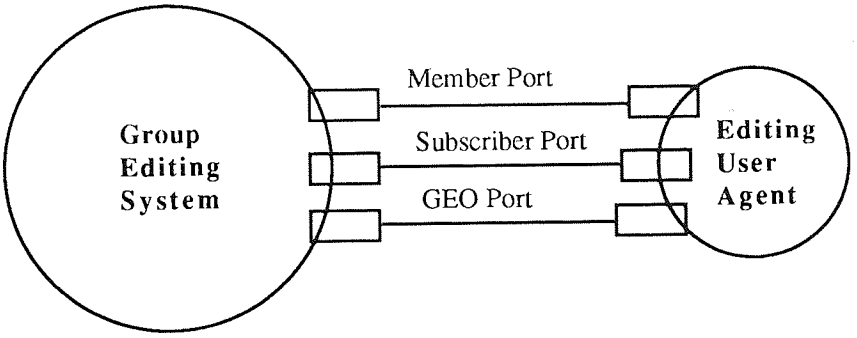


Figure 5.3 Mapping of Communication Ports with CCITT Draft.

5.5.3 Access Control

Access control (Smith et.al., 89) is the need for privacy and integrity of information where information is shared among many users in a distributed network environment. Privacy prevents people from retrieving unauthorised information or allows them to hide information. Privacy is required when dealing with personal or sensitive information. Integrity aims to provide correct and meaningful information. Access control assists integrity by ensuring that only authorised persons are able to maintain the information. Access control grants permission to perform operations. The operation available vary from application to application and who operates them. The operations must provide the functionality required by a specified operation. For example, '**update-contribution**' operation should be able to meet the functional requirement involve in it. The draft X.gc (Joint ISO/IEC/CCITT, 91) on group communication system, contains abstract service definitions that specify the following operations allowed to a group while it is in operation (a complete abstract service definitions of these operations are given in Appendix 4).

Initial operations which need access support for each role in the activity are given below (Joint ISO/IEC/CCITT, 91):

i) create an object with distinguished name and attributes --> **create**,

ii) delete a specific object --> **delete**,

iii) provide read mechanism for retrieving information for a single object --> **read**,

iv) modify of an attribute belonging to a single object --> **modify**,

v) establish relationship between objects --> **links**,

vi) remove relations --> **unlink**,

vii) specify link to the named items --> **linked-to**,

viii) name all items to which named items has the specific link --> **linked-by**,

ix) find items in defined area -->**search**.

All these operations assign rights for manipulating a specific item (i.e. read, modify, delete etc.). The right however, to create a new activity or add/delete a member or allocate rights to other members is given from a higher level, superior node. In case of group editing activities this right is with the group editing organiser.

The access control mechanism requires a method of describing users so that they can be assigned access rights within the application. As the directory service maintains the users directory names, the users can be represented with their directory names (Smith et.al., 89). The names of subtrees in the Directory Information Tree (DIT) could be used to represent groups of users, which would also allow access rights to be assigned on an organizational basis as reflected by a directory information tree.

5.5.4 Storage Facility

The CCITT X.400 series of recommendations 1988 included a new entity: message store. Each message store is to be associated with a user agent, but the reverse is not necessary. This is to meet the storage requirement of remote networked workstations on a single user basis. The requirement for a group user to store common contributions is to be met by having a common storage facility serving as common store or common database. Palme (Palme, 87 & 88) considers that conferencing should have a distributed common database for storage of group contributions. The AMIGO report (Smith et.al.,

89) considers that the Group Communication Services Information Model should support distributed storage to avoid communication overheads in a very large distributed environment, having different molecular structure, such as chains, trees or nets. The issues of common store are concerned with locating objects, replicating objects, distributed update and consistency of information (Nunez, 89). The Grace conceptual model considers that the structure of various group communication objects are rather like a data dictionary that defines the structure of information in a database.

The AMIGO internal information model identifies the requirements which must be met by the internal data model (Nunez, 89). These include representation of basic communication objects, additional information needed for distribution handling, elements and operations to support the activity, mechanism to preserve information integrity, multi-user characteristics and global naming structure.

The services provided by an information model (which forms a common store) are described in section 4.1.1, chapter 4). Group Editing activities have a different structure of information as they have more than one versions on a same topic. The description of the common store is one of the major issues on which work is being carried out at other places. The structure of common store is not an issue in the 'SAGE' project. However, the concept used to define common store in the 'SAGE' project is that there is a common work place for all the group members which is centrally located (to be defined by GEO). This common store has a nature of distributing information (from a centrally located storage) to its members on a needs basis and links the new versions to the rest of the members (i.e. other than the originator). This distribution of information is handled by rule-based information handling system. The retrieval and the storage requirement to and from the common store is dealt by the rule-based information handling system.

Two different types of common store structure are discussed in the 'SAGE' project during analysis and prototyping the group editing activities. The first type of structure is discussed precisely in this chapter, and has a set file structure. The same structure is considered for the Newsletter prototype implementation. Another type of structure, which is a single folder, is discussed with the technical paper.

The proposed structure (discussed here) for this project is that, there would be a centrally located common storage facility to store common contributions. The common store is basically divided into two logical parts. The first part is for the contributions from all group members and the second part is for group editing organiser (GEO) to accumulate accepted versions and to allow document handling. The first part has separate individual folders for each member of the group which contains their individual contributions. The second part is a set of folders which are defined for handling of accepted versions, request, suggestions and reminders.

Contributions arriving for the GEO are temporarily kept in the MH-Directory (Message Handler-Directory), and are transferred to an appropriate part of the common store when they have passed through the monitoring system. The rule-based information handling system should be able to retrieve these contributions for a given operation. For example, a read operation will display the contribution, to be retrieved by an appropriate group service agent.

The group editing organiser would control the access facilities (read, write, edit, and delete), for contributions in common store. The group editing organiser may however, assign various operations to the other members of the group.

5.5.5 Version Control

The version control mechanism varies from application to application and is a major issue in group editing activities because at any moment there may be a new version. The aims of controlling versions are to avoid redundant editing, ensure distribution of an appropriate document and retrieval of old contributions, and also to save disk space.

There are many ways to deal with version control problems, for example: locking, dynamic editing, page/screen editing and/or last version tracking. The approach used in 'SAGE' project is similar to the CCITT draft X.gc (Joint ISO/IEC/CCITT, 91), where each single edited contribution is a separate node point having its own identity. There should be a tracking mechanism for individual's contribution, which always retrieves the latest and highest versions at a base level. The base version (from where the tracking is required) would form a **derived link** (Benford & Palme, 93) in the structure of information model. All contributions should be available to all the members of the group as read only, but originator and GEO have extra access rights to a contribution. If an intermediate version is updated, it will become the latest version in the common store.

The 'SAGE' project introduces the concept of edit request to avoid redundant editing. The redundant editing is to be controlled by access rights provided to the members. The edit request has to pass through these access rights for each member. If a topic has more than two authors or a document itself is edited by more than two people (e.g. a technical paper) and they want to edit the same contribution at one time, then an editing request has to be made. This edit request requisition would be approved by the application service agent (i.e. by rule-based information handling system), which is monitoring the activity. If the contribution (which includes all versions) is not being updated by any member then it would be released for next update, and locked as 'not to be updated' until the arrival of next new version. The request for edit can be made for any contribution in the common store. The proposed concept is contrary to the technique

described in AMIGO report (Nunez, 89) where next update can only be applied on the latest version which is considered as a master document. The reason for considering all versions for the next update is to give extra flexibility to the author. This is possible because all previous contributions are available in the common store.

5.5.6 Message Flow

The message flow within a group editing depends on the activity and the originator. Depending on the communication set up, the message flow takes the shape of a one to many, many to one or many to many communicating environment. For example for a bulletin board, the set up is formed for one person to many, whereas for editing a paper, the set up is formed for many to many. An originator may send a message to selective members of the group. The Group Editing Organiser may send a message to the referee editor who may be invisible to other members of the group. In some cases a referee may be able to deliver messages direct to the author concerned. If a message is originated by Group Editing Organiser, it may be sent in broadcast mode. The GEO should be able to decide the frequency of all message traffic (e.g to reduce communication overload). Messages (each one is considered as a separate node) may be bundled together which may economise the communication cost in case of long geographical links. In the case of notification (e.g. delivery notification) (Joint ISO/IEC/CCITT, 91) a message can be originated by a process (built-in optional facilities) according to the requirements of the activity concerned. The particular mechanism of message flow depends on the activity being carried out.

5.5.7 General Facilities

While defining standards and user tools there is a requirement to try to ensure that the requirements of a large number of users are met. But the system should not have excessive overhead caused by meeting minor requirements. System features like direct

communication between group members (in context of group editing), need of notifications & reminders and role of referee would be set as a default options. System features may be divided within two categories: controllable by a group editing organiser and controllable by other members of the group.

The message handling system is a store-and-forward system. That means a handshake mechanism is not required. A delivery failure checking mechanism should be provided, and if there is a delivery failure the contribution should be searched and re-sent to the destination. The editing system should also be supported by automatic reminder facilities with reminders sent to members who have been inactive for a set of time.

This chapter has discussed the need for a case study, and the method used for the case study, in the 'SAGE' project. The nature of group activities suitable for case study has been discussed. From these considerations a basis for a conceptual modelling is outlined. The next chapter applies the conceptual model to the specific task of editing a Newsletter.

Chapter 6

Editing A Newsletter: Prototype Specification

6.0 Introduction

The Newsletter editing activity was selected as the main case study topic as described in chapter 1 (section 1.4.2). This chapter discusses the issues that arise in the group editing of a Newsletter in a networked environment, making use of the literature and of user interviews. The Newsletter editing activity is developed on the basis of the general model described in previous chapter, proceeding from requirements through progressive design steps to a working prototype of a Newsletter editing system.

A breakdown of the Newsletter editing activity is necessary to develop the requirement definition for the implementation. An initial prototype was implemented to test the system on potential users and to support and develop ideas for the research. The user testing sought to discover any misunderstanding of, or omission from, the user's requirement. User testing and personal communication may suggest further requirements.

An analysis of a typical Newsletter activity (section 6.2) was used to develop the requirements for the prototype. The prototyping activity was designed to test the requirement of underlying functionality demanded by the message handling system and group editing tools. The case study does not address the requirements of the user interface. Such requirements are of course of the greatest significance in a complete implementation, but the 'SAGE project research relates to necessary extensions to X.400 and its services, rather then the human interface side.

Obtaining the necessary extensions to X.400 is achieved by designing a throw-away prototype. The prototype is developed by system modelling, which allows extraction of additional requirements, and implementation of the system as a prototype. The prototype provided the foundations for the recommendations for group editing support services tools which are additions and/or modifications to X.400 as defined in the group communication services draft recommendations X.gc (Joint ISO/IEC/CCITT, 91).

6.1 Selecting the Prototype Technique

Implementation of the prototype is a software engineering problem. Software engineering produce a number of techniques to design systems. Four of these were considered for use in this work: the waterfall method, the exploratory programming method, the prototyping method and the formal transformation method (Sommerville, 89) are all widely used. The first three are more widely used for practical system development (Sommerville, 89). The exploratory programming and prototyping methods are similar in nature and involve developing a basic working system. One of the important functions of each design technique is to establish the system requirements. The exploratory programming method is useful in the early development of new, large and complex systems. Exploratory programming produces an incomplete system which is developed from an initial incomplete understanding of the requirement and progressively augments the system as new requirements are brought out from testing of the initial system. Exploratory Programming methods tend to result in a system whose structure is not well defined and therefore hard to maintain (Sommerville, 89). The benefits (Sommerville, 89) of using prototyping during the requirement analysis and definition phase are awareness of the users requirement, detection of missing user services and identification and refinement of confusing services.

The case studies, literature and user interaction provided sufficient requirements information to go directly to the prototyping method. The initial phase of the study

comprised collecting potential user's comments on functionality and reviewing the literature. The results of these two phases were used to build 'good' specifications. These specifications were used to build a prototype as a prelude to application of prototyping methods which could improve the specification requirements.

There are two different types of technique within the prototyping method. One is the 'throw away' technique which does not really concern the implementation part. The other leads to the implementation as a closed loop which involves validation, verifications and refinements. The throw away technique is used in this research. Maintenance is not an issue for a prototype.

The prototype is mapped onto X.400 in the context of the group communication services and this leads on to the review of the case study. A graphical representation of the development phase of the research is shown in Figure 6.1.

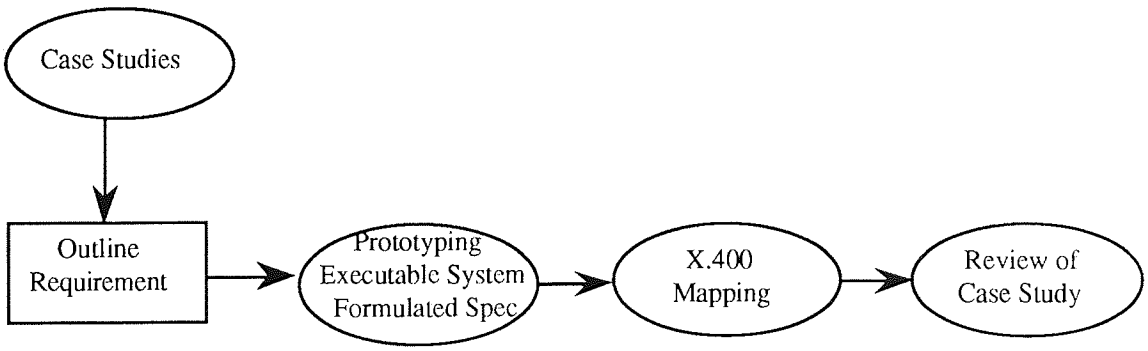


Figure 6.1 The Development Phase of the Research.

6.2 Editing a Newsletter: Analysis

In the context of group communication, the system for producing Newsletters is an environment which contains a tool set for production of a Newsletter in an OSI network environment. The environment allows contributions from multiple authors at various

levels of editing, browsing and co-ordination. The activity is controlled by the group editing organiser and consists of three basic group communication classes related to the CCITT draft: entity (members), item (contributions) and domain (geographical range of network i.e. OSI boundaries) as shown in figure 6.2. An example of group editing of a publication is the production of the fortnightly Newsletter 'ASTON FORTNIGHT' in Aston University. The domain and/or period of publication may be extended or changed for such Newsletters, say a monthly publication for all Universities in UK or Europe.

The CCITT draft X.gc (Joint ISO/IEC/CCITT, 91) specifies the group communication information model and the abstract service definition to provide a framework for modelling and implementing a variety of group communication applications. The information model for editing a Newsletter consist of three base classes of objects. The items which take part in the process are contributions, subjects, topics, the Newsletter document and parts of the document. The entities are author(s), co-editor(s), co-ordinator(s), reader(s) and agents. The domain of the activity is limited by the extent of the OSI network.

The block diagram in Figure 6.2 shows the scope of the Newsletter information model. For example: domain A and domain B are two different organisations in the country X and domain C is another organisation in the country Y. These organisations produce a technical Newsletter for which the activities are controlled by the group editing organiser at domain A. However, the Newsletter is produced by the authors, editors and co-ordinators spread over the countries X and Y in all three domains, using the services of the message transfer agent.

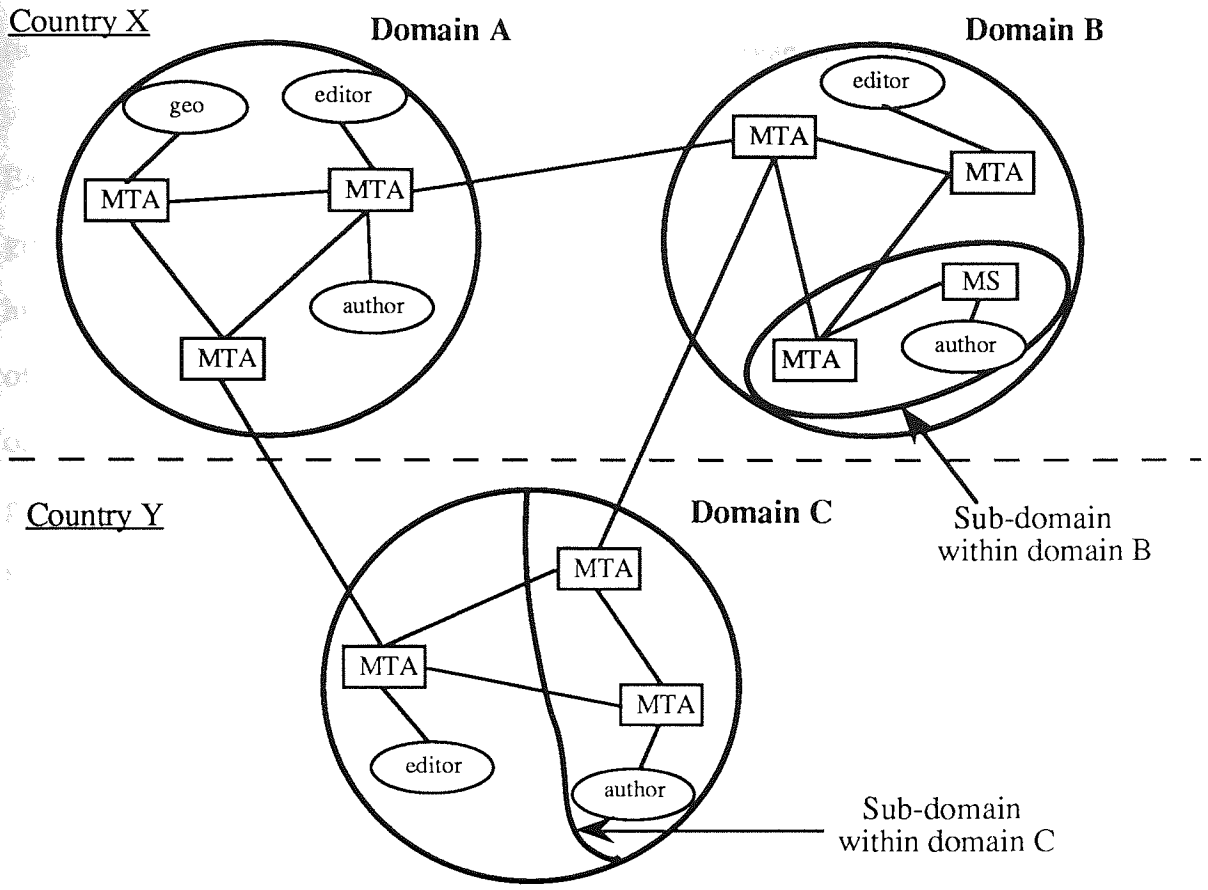


Figure 6.2 Scope of Group Editing Information Model.

6.2.1 Structure of the Newsletter Editing Activity

The structure of a group editing activity information model comprises communication between members and the common store via operations provided to meet the requirement of an activity. Initially the operations (defined in section 5.5.3) are considered to be the same as specified in the CCITT draft release. These operations are to be constructed to meet the requirement of the editing group. The operations are grouped (Joint ISO/IEC/CCITT, 91) into three communication ports. The communication ports are modelled onto the editing Newsletter activity as below (discussed in section 5.5.2):

(i) Member port ---> author(s), referee or guest editor,

(ii) Moderator port ---> co-editor(s), co-ordinator(s) and

(iii) Conferencing Manager port ---> group editing organiser (GEO).

Each port should specify the access control operation to the concerned member of the group. These three ports form a hierarchy of operations in terms of access right. The hierarchy of operations is that the operations for a member are a subset of the operations of co-editor/co-ordinator. The operations of co-editor/co-ordinator are a subset of operations of the group editing organiser. The access right hierarchy structure of the group is shown in figure 6.3. The member level has access to least operations and the subscriber level has access to more operations than a member. The moderator level has access to all operations.

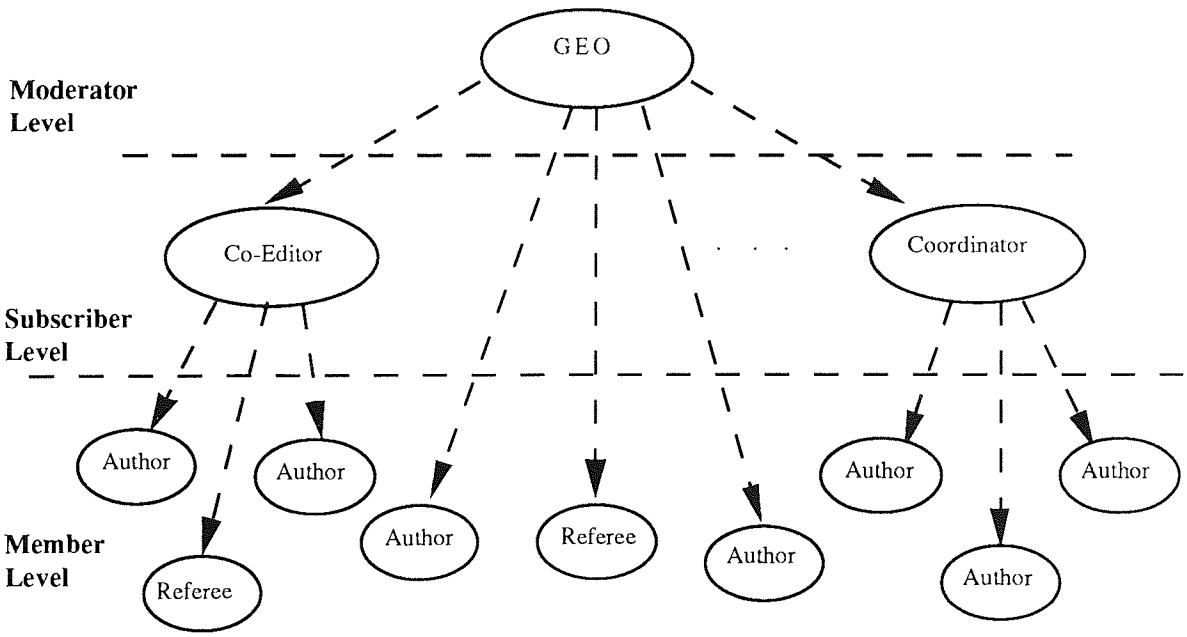


Figure 6.3 Group Hierarchy Structure.

6.2.2 Working on Newsletter

This section discusses the basic requirement and message flow mechanism of the Newsletter editing activity. It is considered that the activity consist of the object classes as explained in section 5.5 and the access rights hierarchy (similar to draft X.gc) shown in Figure 6.3. Initially the group editing organiser has to prepare a list of topics/themes

in relation to the Newsletter. The GEO also has to decide the other members of the group. The group editing organiser may however communicate with such member(s) before creation of the group by means of the normal services of the message handling services, for example.

6.2.2.1 Creation of Group Activity

The create operation is described in CCITT draft X.gc (Joint ISO/IEC/CCITT, 91). Only the underlying functionality involved during the create process is discussed here. The group editing activity may be created by any user by taking four basic measures to initiate the activity process in regards to group administration. The originator of the activity is known as the group editing organiser (owner) for the activity. The group editing organiser is the only one empowered to access the operations described below.

(i) Definition of activity -- group editing organiser should be able to define the name of the activity which would be access by this name in future (e.g. name of Newsletter).

(ii) Definition of group -- group editing organiser should be able to define group members with their status such as: author, co-editor or co-ordinator.

(iii) Definition of Common store -- The common store structure is created by the Group Editing Organiser (the GEO decides where to create the common store).

(iv) Definition of activity history -- activity history database is also defined by the Group Editing Organiser. It contains complete information about the activity/application, group and storage definitions and other informations such as: date of creation, critical date and topics with their associate members.

Figure 6.4 shows the structure of the database in relation to the activity concerned. The definition of activity should be under an application group which would be useful to

identify the group application type for a wide range of activities (discussed in section 4.3 chapter 4).

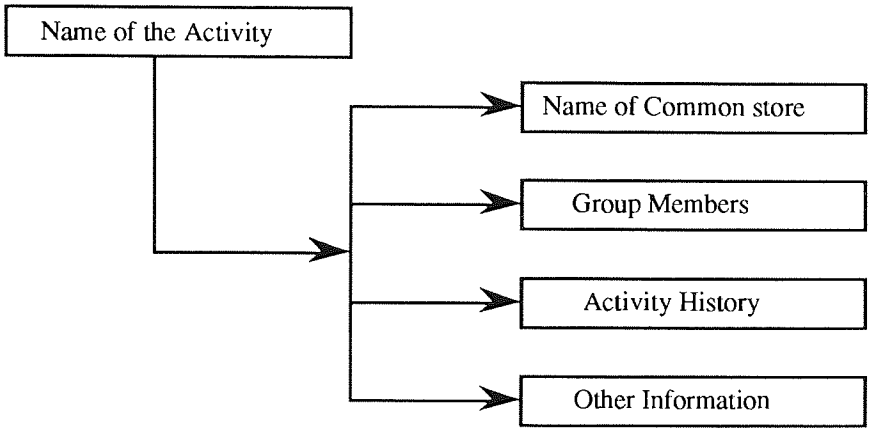


Figure 6.4 Structure of the Activity Database.

6.2.2.2 Initial Message flow

The activity, which has been defined and which has come into operation has its status maintained until it closes. The group editing organiser has to send Newsletter information (subject, topic etc.) to author(s), co-editor(s) and co-ordinator(s) in the form of notifications to each member of the group. Figure 6.5 shows the flow of initial /messages/contributions. The author(s) and co-ordinator(s) may accept or reject or ask for modification. In case of rejection no action is required by author(s). The Group Editing Organiser may however, exercise the same action with another member (author). In the case of acceptance the author has to write the news and co-ordinator/co-editor has to distribute the topic between author(s). In the case of modification, author/co-ordinator may suggest/modify the issues and send it to the Group Editing Organiser. This process is repeated until both agree. The modification may include issues such as completion date and payments.

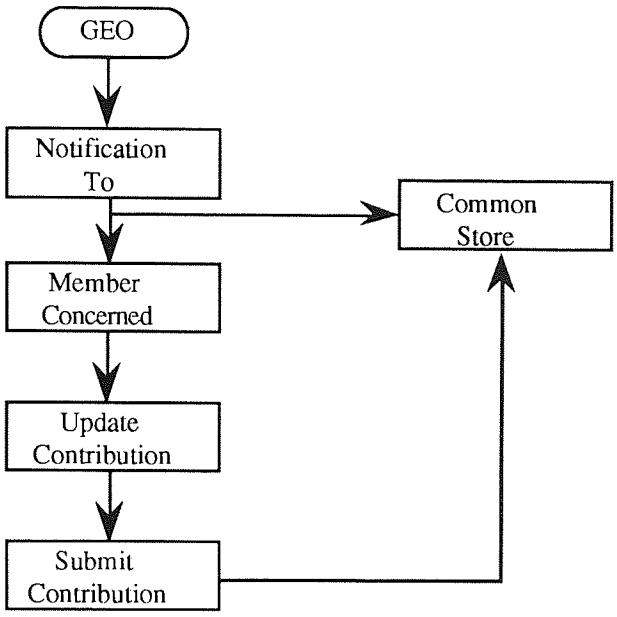


Figure 6.5 Initial Contribution Flow.

6.2.2.3 Contribution Flow

Each contribution in the common store has a unique identity and follows the X.400 messaging format. Most of the time the contribution flows between a group member and the group editing organiser. Each update by a member submits a new contribution to the group editing organiser and is available in the common store, as read only, to all members of the group. An edit request can be made by any member to update their own contribution. The request is based on subject, topic and contribution number. The request flows from requesting member to the group editing organiser. The released version flows from common store to the member concerned.

A suggestion can also be made by any member of the group. This flows from the suggesting member to the group editing organiser for record, and a copy of the contribution containing the suggestion flows from the group editing organiser(common store) to the originator of the contribution. Figure 6.6 illustrate the contribution flow between group editing organiser and member concerned.

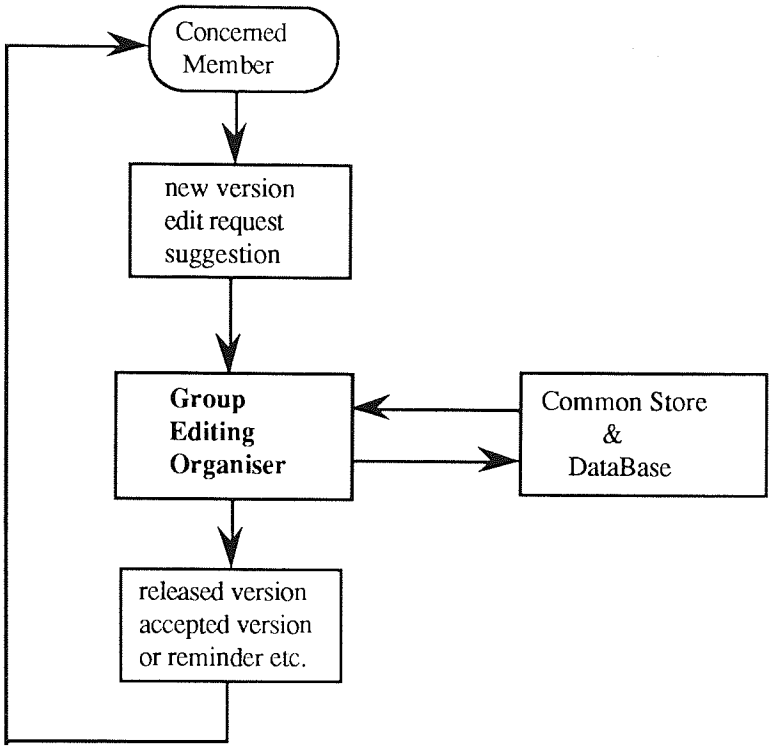


Figure 6.6 Flow of Contribution During Operation.

If the contribution is related to a technical issue then the group editing organiser may send the contribution to a referee (guest-editor) for comments. The group editing organiser may wait for similar contributions to bundle these which are to be sent to one referee. The commented contributions may directly be sent to the originator or it may be sent via group editing organiser.

6.2.2.4 Group Editing Header

The group editing header is defined as an extension fields of IPMS (X.420) in context of X.400 messaging format. The header parameters will be displayed with each editing session. It is proposed the group editing header comprise the following parameters:

i) Reference of News Letter : ' name of the news letter '

ii) Status of the Member: 'author/editor/co-ordinator/reader'

iii) Current Topic : ' topic for concerned author '

iv) From GEO/Co-ordinator/Co--editor : ' from whom contribution is sent '

v) To Author/Co-ordinator/Co--editor : ' to whom contribution is sent '

vi) Latest Submission Date : ' applies to author '

vii) Updated on : ' date ' (optional)

viii) Part of Newsletter : ' Newsletter to which it belongs '

ix) Revision Suggested By : ' Suggesting user member ' (optional)

x) Version: 'Released for Edit' (optional)

xi) Version: 'Accepted' (optional)

Some of the header components (e.g. from and to) are defined in X.420. The 'SAGE' project proposes the header components at number i), ii), iii), vi), x) and xi). These header components are required to meet the functionality involved with the activity.

The optional header parameters are visible when such fields are required. For example, when a version is released for next update, the parameter 'version released for edit' is visible. Since all the members will have read only access to all contributions, any member of the group can suggest change to any contribution. In such case 'Revision Suggested By' parameter is visible along with other parameters of the header.

6.2.2.5 Editing a Contribution

Any group member with access rights beyond read only can update an existing contribution or create the first draft after receiving the first notification from the group editing organiser. The contributions contributed by a member, or updated by group editing organiser for a member are available in the common store. To update a contribution (other than first draft) a member has to make an edit request to get the required version from common store. The contribution which is released from common store for update carries an additional header parameter 'Version: released for edit'. After

editing, if the version is submitted, it is available in common store as the highest version.

The group editing organiser can edit, without making an edit request, any contribution at any time, provided the contribution is available for update, that is, the contribution is not currently open for updating. The system should check this for the group editing organiser.

The common store holds all contributions for all members within their allocated folders. The group editing organiser can accept the contribution send by any member. If a contribution is accepted, it should disable further editing the author member on that particular subject and topic. The accepted contribution is available in the document folder in the common store and carries an additional header component 'Version: Accepted'. The copy of the accepted contribution is sent to the originator.

Any member other than the originator can send a suggestion for an alteration to the work of another member. In the case of a suggestion, the suggestion is appended to the original contribution, along with an additional header parameter 'Revision Suggested By'. A copy of this suggested contribution is sent to the originator of the contribution. This is available in the request folder in the common store. The group editing organiser or originator of the contribution may act on that contribution according to the suggestions and may edit the contribution to take account of the suggestion.

The group editing organiser may ask the referee (or a guest-editor) to comment on any particular contribution. In such case the referee should be able to edit that particular version of the contribution. Alternatively the group editing organiser may ask the referee to comment in the form of a suggestion, not allowing updating of the contribution. The

comments on a contribution from the referee will only be available to the group editing organiser, who may pass it to the author.

Based on the submission date for date critical topics and subject, an automatic reminder should be generated by the system to the concerned member if it is required.

6.2.2.6 Retrieval of Newsletter

As described in the section 6.2.2.5, the accepted versions of contributions from each group member are available in the document folder within the common store. To retrieve the Newsletter document the sequence of the contributions is required. The Newsletter document will appear in the order according to the selected sequence number. The Newsletter is held in the home directory of the group editing organiser. Different versions of the Newsletter can be generated by changing the order of the sequence of accepted contributions. The Newsletter can then be submitted for publication and/or automatic delivery to any or all members of the group. Only the group editing organiser is empowered to submit the Newsletter for publication.

The individual contributions can also be retrieved on a selective basis by using 'mhtools' (Peek, 91). A 'pick' command can retrieve the contributions based on subject, topic, member and date etc. This is very useful at later stage for a complex structure of information model (Joint ISO/IEC/CCITT, 91) which needs special searches for information handling.

6.2.2.7 Other Facilities

The approach adopted for delivery notification is the same as in X.400 (i.e. non-delivery notification). If the contribution is not delivered at its destination, the non-delivery notification (X.400, 88) is delivered to the originator as delivery failure. The

system would be able to check automatically such non-delivery failure and the contribution re-sent to the destinations.

The system would have a notification facility after receiving the first notification from the group editing organiser. The first notification is the registration to be a member of the group.

6.3 Implementation Specification for the Prototype

The important features of prototyping is to establish the requirement specifications. A throw away prototype is developed in this research. The prototype is the testbed for users views and suggested ideas.

6.3.1 Functional Requirement

This prototype deals with the various aspects of handling of editing by a group of people working together for a Newsletter. The group is assumed to be operating in an open systems network environment. The functional or specification requirement defined here is basically the solution of the activity described in the previous section 6.2 which tries to determine what the users want or what services should be provided for the users. This requirement is to be concluded as group communication system services tools for editing Newsletter.

The problems of group editing (updating, co-ordination and access etc.) are described in section 5.4 which explores group editing issues for editing activities in general. The requirement described in the analysis of the Newsletter activity (section 6.2) is a key tool for prototype system mapping on X.400.

The entity description in table 6.1 may also be referred to as a top level requirement definition. This requirement may be divided in to three parts. The first part describes the

initialising requirements for the activity. The second part covers the supplementary requirements which are those activities needed during the operational phase. The third covers is the monitoring activities needed to update the status of the activity after every time interval.

Name	Description
Activity Monitor	Rule-based Contribution handling system e.g. handling of edit request, suggestions, reminder, delivery failure and new contributions.
Activity Generator	Menu based system used by the group editing organiser to create activity, common store, group and to work with contributions in the common store.
Activity Responder	Menu based system with member concerned to make edit request, update contribution, and make suggestion for other member's contributions.
X.400 (88) Implementation	Use of X.400 set rules and extension of the P2 header to get editing header parameters; part-of-news-letter, updated-on and latest-submission-date.

Table 6.1 Entity Description.

6.3.1.1 Basic Requirement

The basic requirements are initial needs required to operate the activity:

i) creation of activity (Newsletter) database,

ii) creation of a common work place for group contributions,

iii) creation of a group members list to identify those working with the activity (with their status; viz author, reader, editor and co-ordinator),

iv) creation of a header for a particular operation (like edit, accept and suggest),

v) generation of initial notifications to the concerned member.

vi) edit contribution,

vii) accept contribution and

viii) generate Newsletter in a desired order.

6.3.1.2 Optional Requirement

This requirement may be viewed as an optional requirement. It is not necessary that all of the requirement be used while the activity is operating:

i) add and delete member,

ii) make an edit request to get a copy of a contribution from the common store,

iii) make a suggestion about another member's contribution,

iv) show the difference between two contribution from a member.

6.3.1.3 Monitoring Requirement

This requirement is a rule-based requirement used to update the overall status of the Newsletter activity. The automatic decisions are taken on behalf of the group editing organiser by a set of processes running as a background job over a specific time interval:

i) set initial parameters and update group status,

ii) check for new contribution and process the contribution if required,

iii) check an edit request and release required version from common store if possible,

iv) check for suggestion and process the suggestion if needed,

v) check for reminder, generate and send if necessary,

vi) check for delivery failure, a contribution is re-sent in case of delivery failure,

vii) set read only access for the contributions required to be read by the member's of the group and

6.3.2 System Model

The prototype for the group editing support system models consists of three sub models which operate in parallel. The activity environment is created by modelling three sub-processes on X.400 implementation. The simple block diagram of the model is shown in Figure 6.7 which is supplemented by the entities whose details are described in table 6.1. The notation used here is discussed in Sommerville (Sommerville, 89).

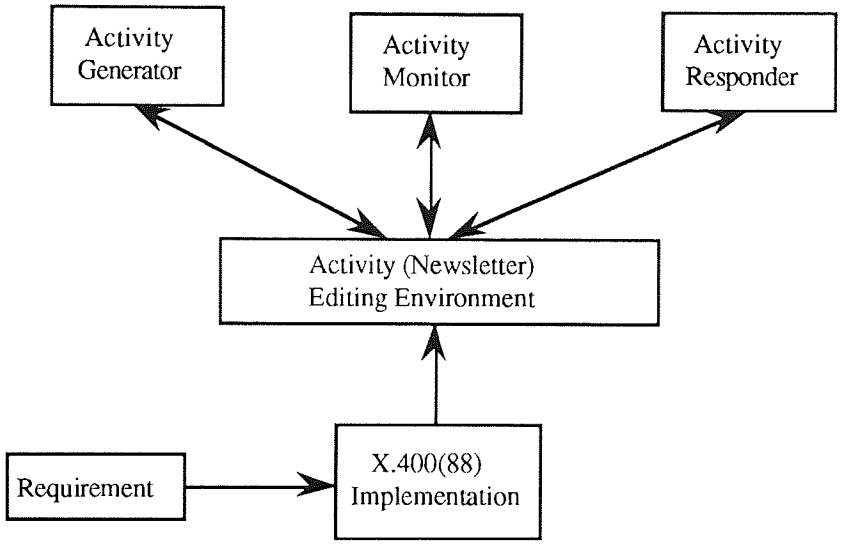


Figure 6.7 Top level Model Activity Editing Environment.

Each sub-system, activity generator, responder and monitor, is independent and runs separately. The activity monitor is a collection of built-in processes which is a rule-based system and runs in the background at pre-specified time intervals (defined by the Group Editing Organiser), shown in Figure 6.8.

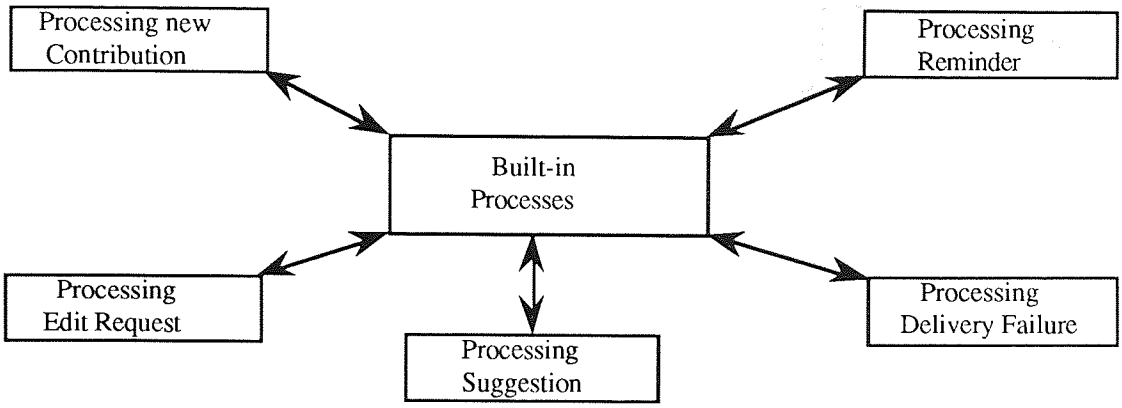


Figure 6.8 Activity Monitor Rule Based Sub-System.

The activity generator is a menu based sub-system shown in Figure 6.9, and requires action by the group editing organiser. The activity generator provides the processing needed by the group editing organiser.

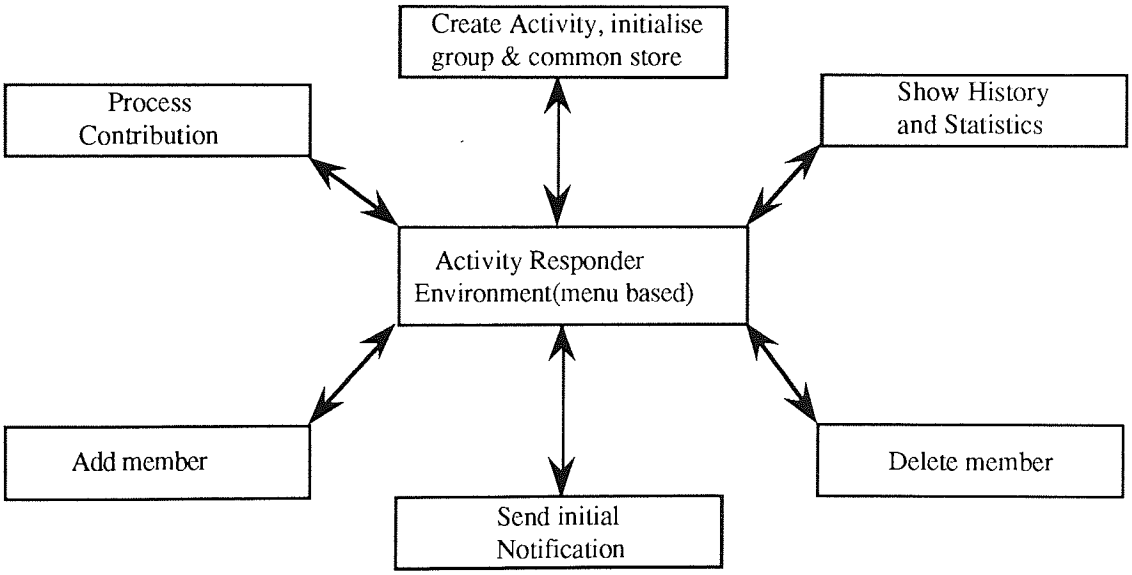


Figure 6.9 Activity Generator.

The activity responder is also a user interactive sub-system (Figure 6.10) which gives a member the facilities to update contributions, provide suggestions for other member's contributions and request a copy from common store for next update along with other functionality.

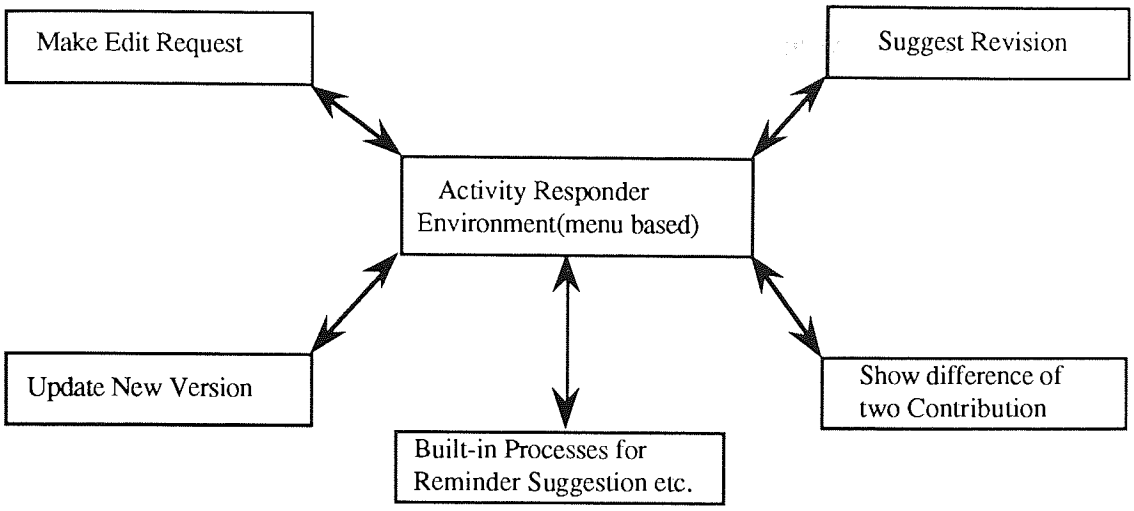


Figure 6.10 Activity Responder.

6.4 Prototype Implementation: Editing Newsletter

X.400 is critical to the prototype to enable it to meet the functional requirement as discussed in section 6.3.1. The X.400 implementation is run under Unix in a message handling environment which supports the Bourne Shell, C-Shell, and C language processor. In the prototype the 'vi' text editor is used to update a contribution. A choice among emacs, promptor or any other editor may be made available at later stage. The current prototype uses user- and host name as a member address. However, the services of X.500 (i.e. CCITT the directory services recommendations) can be used to define group members. In such case the system should support X.500 implementation e.g. the services of QUIPU[1]. But this prototype is modelled over X.400 (i.e. message handling system). This prototype is particularly tuned for the sun sparc2gx work-station, which gives the xmh windowing facilities while editing a contribution on a host.

6.4.1 Prototype Design

The prototyping techniques (discussed in section 6.1) suggests (Sommerville, 89) use of a very high level language, the relaxing of non-functional requirements, ignoring the

[1] this is a X.500 directory services implementation like pp.

considerations of error action and ignoring reliability and program quality standards. The prototyping produces an executable system that may be in an un-structured form. As discussed in section 6.3 the prototyping method used in this research is not concerned with the software design features as it uses the throw away technique.

The strategy used in the production of throw away prototype is Function Oriented Design. The function oriented design is an approach to software design where design is decomposed to a set of interacting units which have clearly defined functions. The approach is used to test the ideas and features for the testbed system.

6.4.2 Data Flow Diagrams

Data flow diagrams are an integral part of design methods and each method uses a slightly different notations. The notation used (Sommerville, 89) here shown in Figure 6.11.

Figure 6.11 Notation Used for Data Flow Diagram.

The group communication editing environment is to be created across the network for an individual activity. For this research project the environment is modelled using three different entities by means of having three independent sub-processes (activity generator, responder and monitor). The entities are group editing organiser, member (viz author, editor, reader and co-ordinator) and built-in process[2] (rule-based system). Therefore the flow of data also takes place to-and-from these entities within the domain of the activity.

[2] a process can be a user and here acts as group editing organiser.

The activity generator sub-process needs the interaction of the group editing organiser to activate the data flow mechanism. A simplified data flow diagram concentrating on functions for activity generator is shown in Figure 6.12.

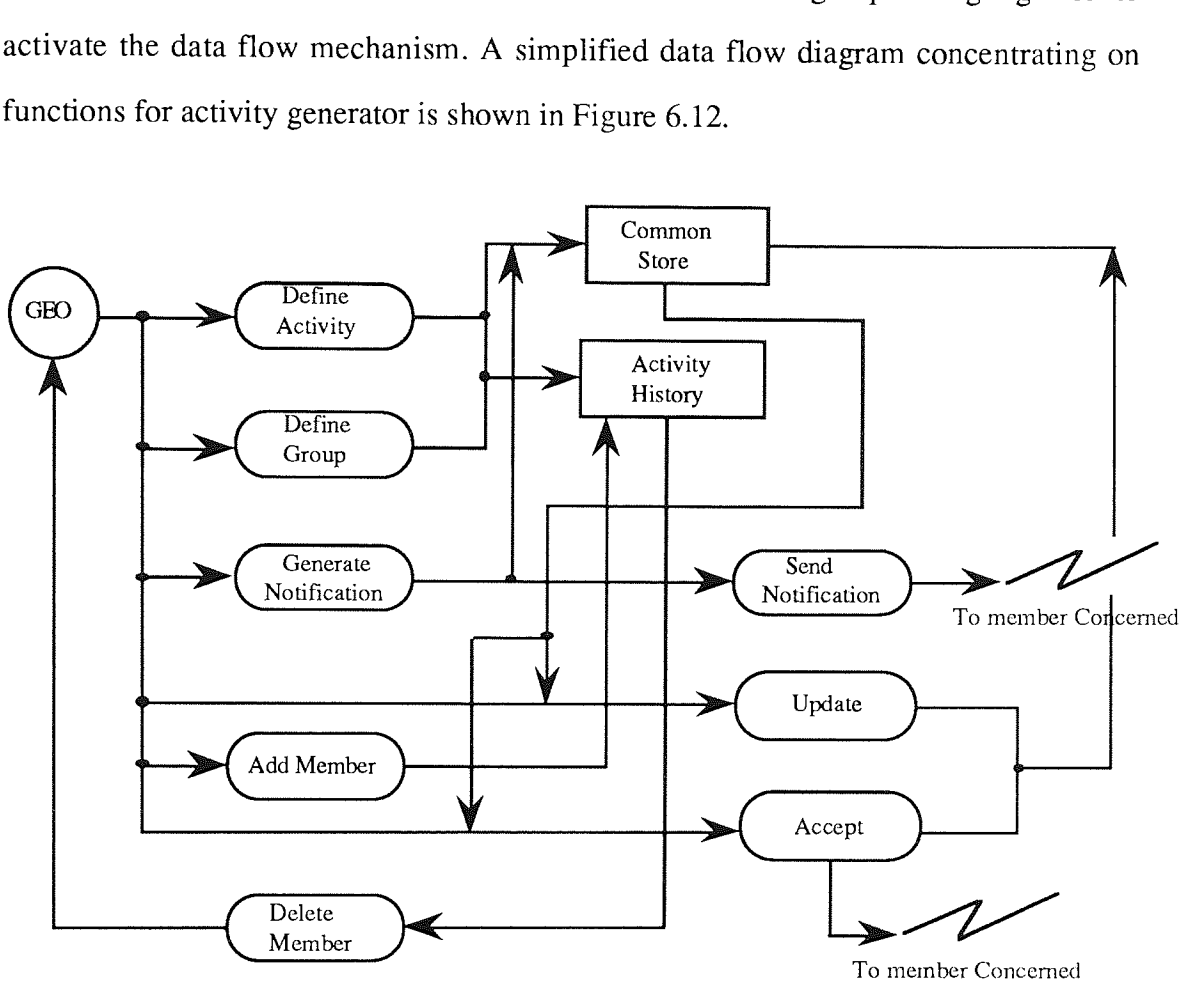


Figure 6.12 Activity Generator Data Flow Diagram.

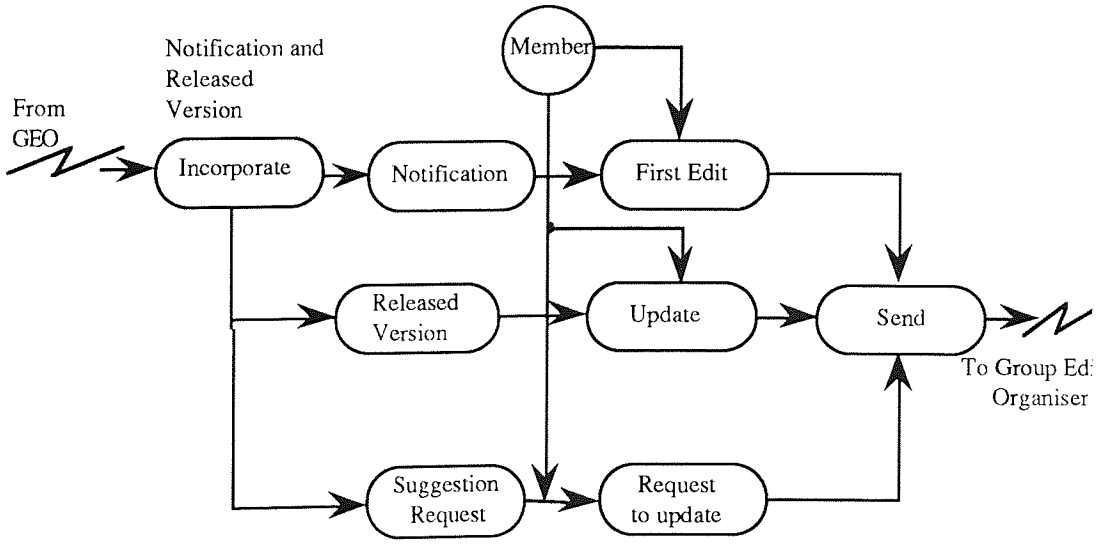


Figure 6.13 Activity Responder Data Flow Diagram.

The activity responder sub-process also needs manual interaction from the member concerned to interact with the activity and the information flows accordingly. Figure 6.13 illustrates a simplified functional data flow diagram for activity responder.

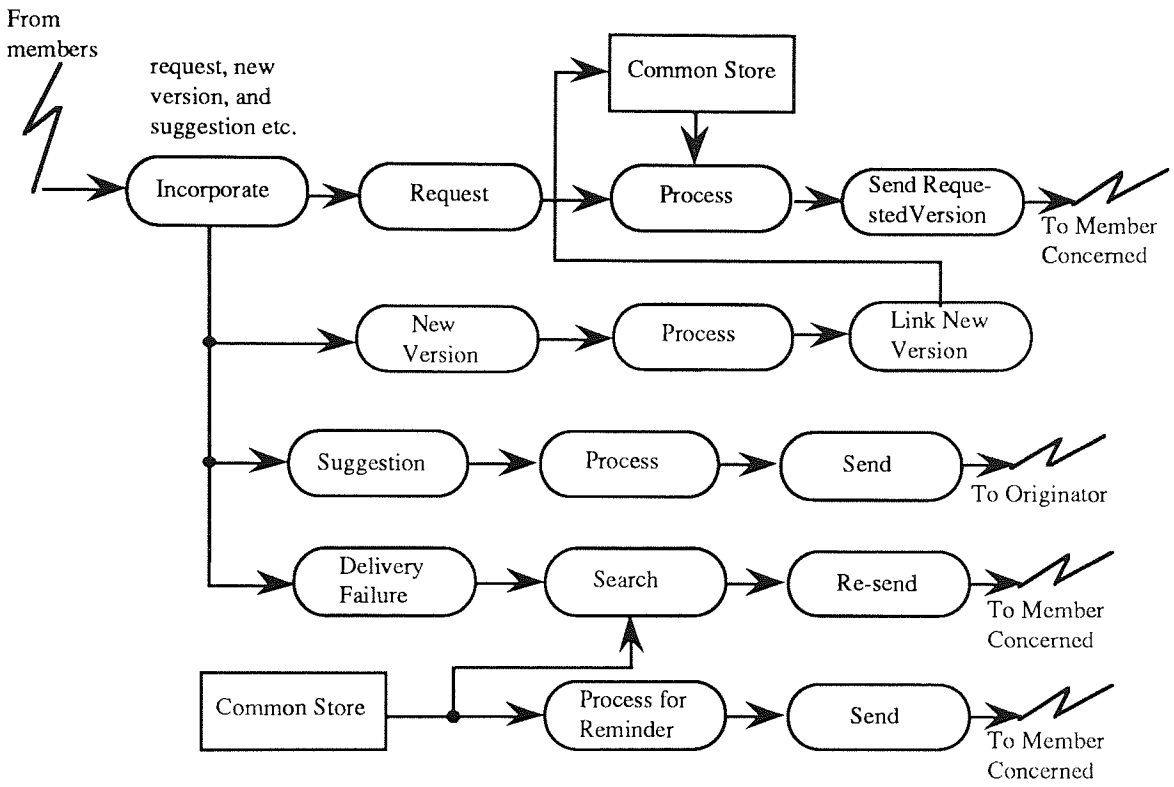


Figure 6.14 Activity Monitor Data Flow Diagram.

The activity monitor sub-process is a rule-based system which acts on behalf of the group editing organiser while running in background. The sub-process takes its own decisions based on the situation (discussed in section 6.3.1.3) and activates the flow of data to-and-from common store and member concerned. Figure 6.14 shows the data flow diagram when the activity monitor is activated.

6.4.3 The Prototype Implementation

As mentioned previously this prototype is developed to test the ideas and acts as a testbed. Logically the Newsletter editing environment is modelled by means of three

sub-systems (activity generator, responder and monitor) which are independent in nature and run separately. Figure 6.15 shows the relationship of these sub-systems. These sub-system are discussed as basic, optional and monitoring requirement in section 6.2.2. Since each sub-system is independent from each other, it is better to describe these separately. The following sections describes the working of these modules. Table 6.2 shows the menu for the activity generator.

The list of activity generator, activity responder and activity monitor is placed in Appendix 5A, 5B and 5C respectively.

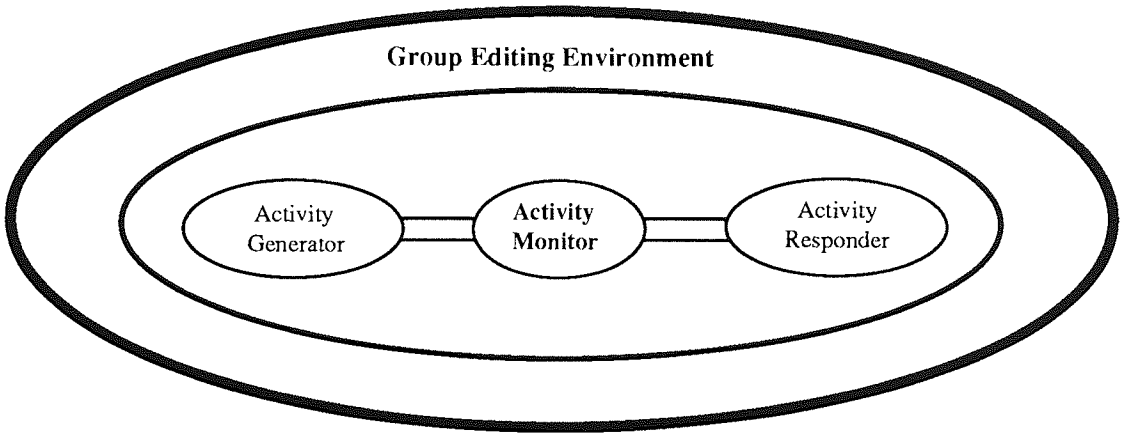


Figure 6.15 Relational View of Editing Sub-Systems.

6.4.3.1 Activity Generator

The activity generator is a collection of various sub-modules developed using C, Bourne and C shell and services of X.400 in the message handling environment. Most of the features of the activity generator meet the basic requirement needed to initiate the Newsletter activity. This is a menu driven module run by the group editing organiser. Table 6.2 show the menu facilities.

Option	Services
A -->	To Create new Newsletter
B -- >	To Add Group Member
C -- >	To Delete Group Member
D -- >	To Send First Notification to the Member
E -- >	To Process Contribution
F -- >	To History and Statistics
G -- >	To Edit Contribution
Q -- >	To Quit

Table 6.2 Menu for Activity Generator.

- **Create new Newsletter**

This is an initialising module for the activity and defines all basic requirement to put the activity in operation. At present the Newsletter editing environment handles only one activity at a time. However, the initialising module checks for the previous activity and if found, warns the group editing organiser about the current existence of Newsletter. If the group editing organiser continues, then closes the previous Newsletter (moves existing Newsletter), advises the group editing organiser for location, and creates a new Newsletter. Figure 6.16 shows the processing of the sub-module (create newsletter) diagrammatically.

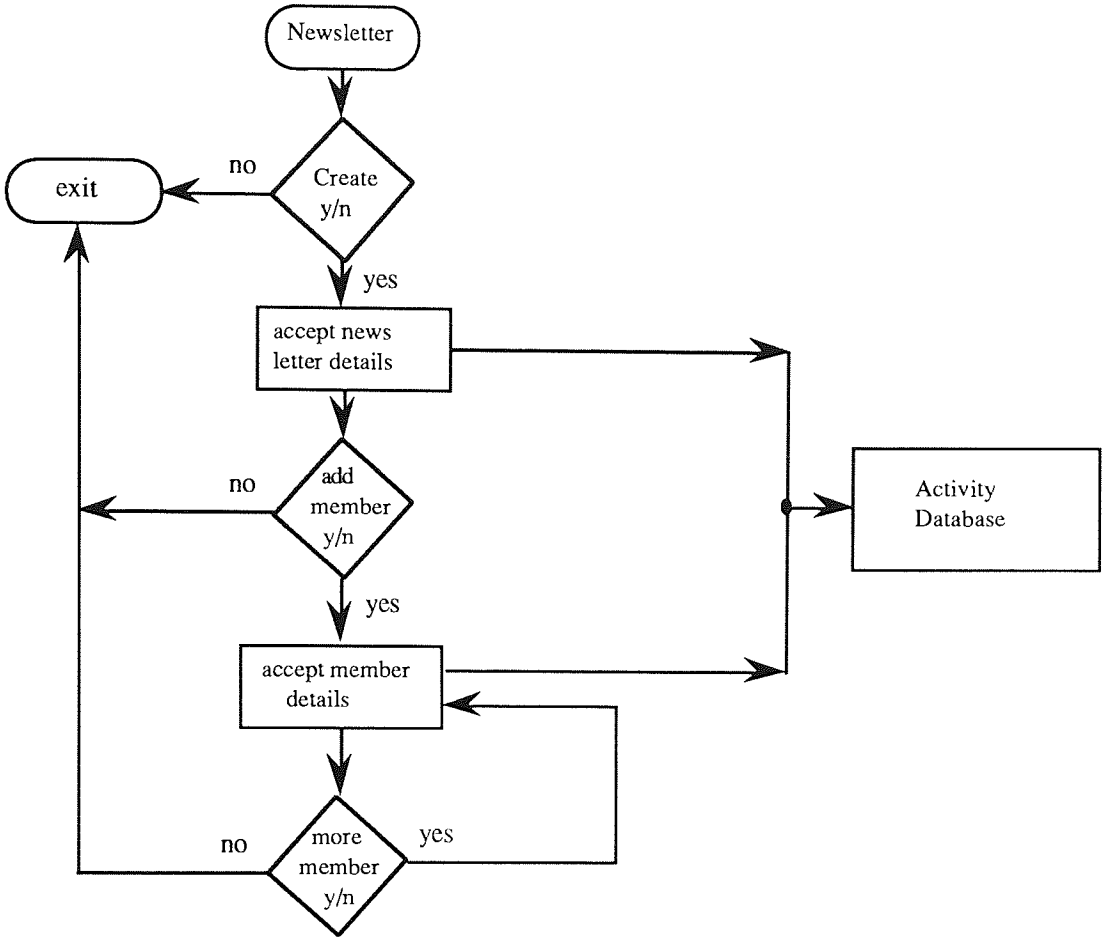


Figure 6.16 Newsletter Creation Process.

- **Add Member**

This module provides the facilities to define the group member's details. It is called by create new Newsletter module if member is included when the activity is initially defined and also called if member to be added later. The sub-module updates the information in the activity database to build-up the historical information. The add member sub-module also makes available the existing contributions in the common store to the newly added member, and creates a place for new member in the working folder within common store. Figure 6.17 illustrate the processing of adding a new member diagrammatically.

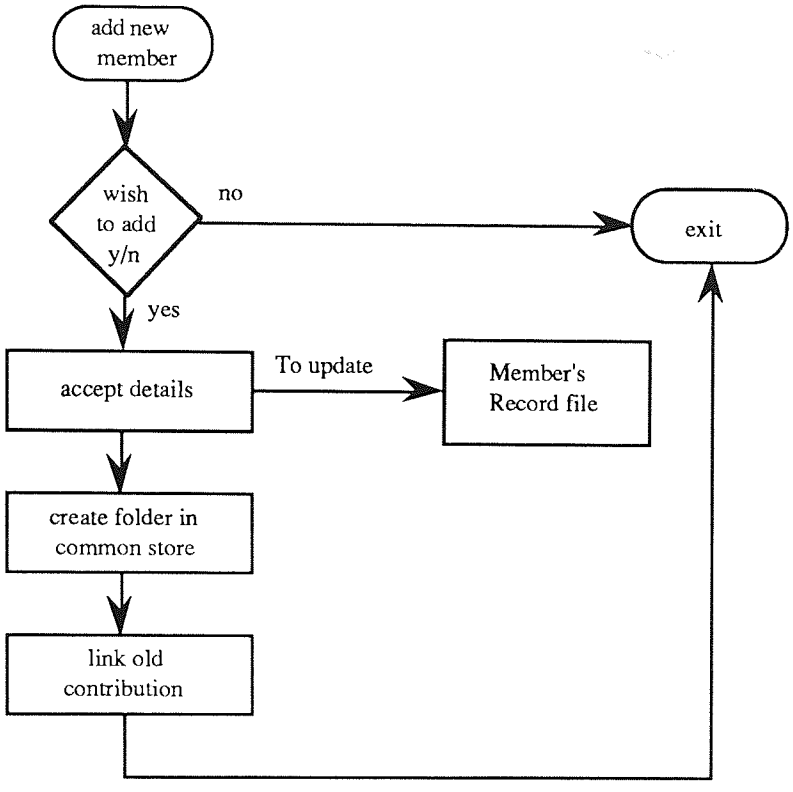


Figure 6.17 Add New Member Process.

There are two different techniques used to access the contributions for the members of the group. The first technique uses the communication link every time when a new contribution is received by the activity monitor, which needs to link it to rest of the members in the group. The second technique accesses the common store as read only once when a member wishes to work with the contributions. The latter technique is used for the networked group member during the testing phase.

- **Delete Member**

This module deletes a member's details from the activity database and disables further interaction between the activity and the deleted member.

The shortcoming of the module is that it does not maintain the record of the members who have been deleted from the group. This might be a useful requirement while

analysing the activity status at later stage. However it preserves the information contributed by the deleted member.

- **Generate Initial Notification**

An initial notification information is the registration of a group member to the editing activity. The activity database is used to construct the header component for initial notification. The module needs to input the address (user name and host name) of the member concerned. The header components consist of the information related to the member viz subject, topic, status, latest submission date and part of the Newsletter. The latest-submission-date, is required to be entered by the group editing organiser which may not be the same for every member in the group. The structure of header is same as a X.400 message, and confirms the P2 protocols header parameter extension facilities of X.420 recommendations. Finally the module enters in to the X.400 message mode and provides complete message composition tools available in the message handler.

- **History and Statistics**

The module formats the information in a required form collecting information from the activity database and from common store. The historical information displayed by the module is: name of the Newsletter, completion date, name of group editing organiser, details of other members and the statistics of the contributions. The statistics includes information about notification, suggestion, and acceptance of contribution for each member. Further information can be added for display as history and statistics. The information considered here, is common for editing activities. This module needs further work to work out the history and statistics in general. However this module is based on an individual activity (i.e. editing Newsletter).

• Process Contribution

This module provides a second level menu facility to the group editing organiser, to work with the various contribution submitted by the members of the group. The second level menu consist of two options, viz accept contribution and generate document.

The first option i.e. accept contribution also needs entry of the group member name and the number of the contribution to be accepted. The accepted contribution carries an additional header component (i.e. '*Version : Accepted*') and a copy of accepted version is sent to the originator of the contribution. The accepted version is filed in the document folder of the common store. Figure 6.18 shows the processing diagram of a contribution by group editing organiser.

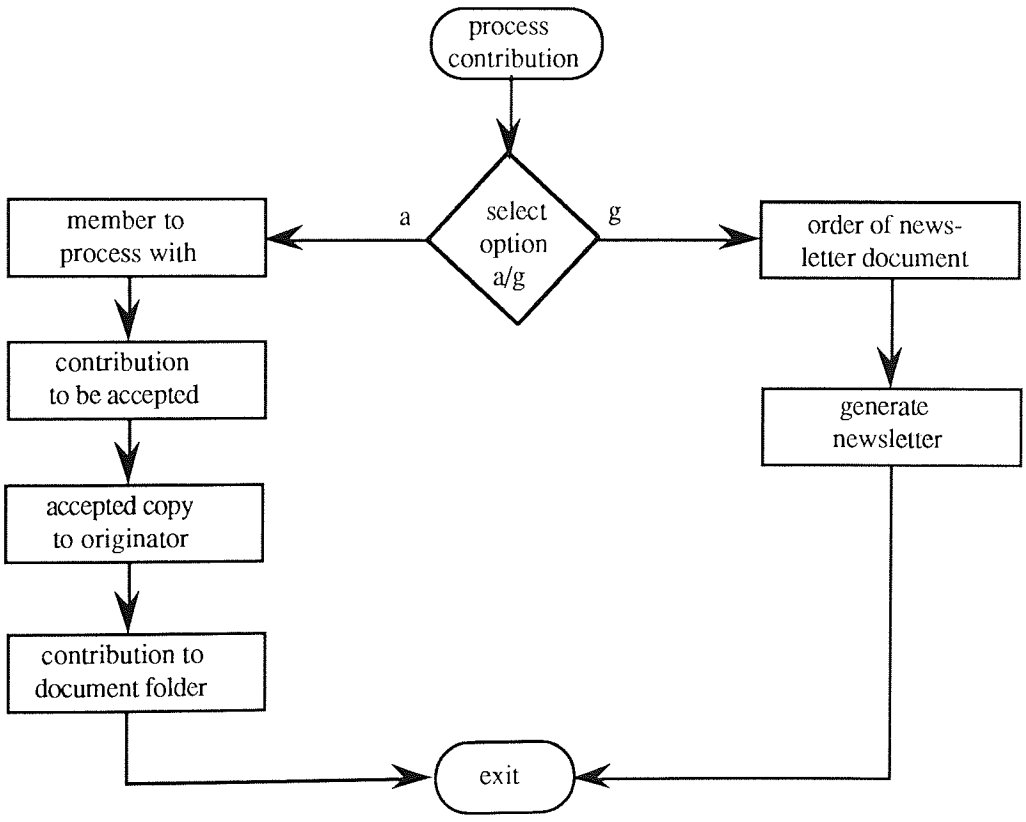


Figure 6.18 Processing a Contribution.

The second option module allows the group editing organiser to generate the complete Newsletter document. The document is generated from the accepted contributions available in the document folder in the common store. The option module requires selection of the sequence order of the Newsletter within the accepted versions. The generated document is made available in the home directory as news.doc. The Newsletter document can be generated many times and at any time.

The structure of the document has flexibility in terms of its presentation. Depending upon the choice and requirement the group editing organiser can reformat the structure of the Newsletter document by changing the format file in Mail directory with the help of any text editor. The format file for the current activity contains the information on subject, topic and author.

- **Edit Contribution**

The edit contribution option module requires input from the group editing organiser. The group editing organiser has to provide the name of group member and contribution number whose contribution is to be edited. If the contribution has been released (by GEO) for editing then the module exits from the menu displaying the message; "Contribution is being updated: wait for new contribution".

If the contribution has not been released for updating the module allows the group editing organiser to update. X.400 message composition tools, such as send, delete, push, edit and refile are available during the edit process. The header parameter of the new contribution is generated from the selected contribution. The updated contribution is to be submitted to the group editing organiser to be a part of the common store.

6.4.3.2 Activity Responder

The activity responder allows a member to respond to a Newsletter activity initiated by the group editing organiser. The activity can be new or old and may be processed by the various operations listed as menu requirement in next paragraph.

The activity responder is also menu driven. The menu facilities shown in table 6.3 are required to meet the functional requirement of this module. Apart from the menu-based facilities, the activity responder also does the initial checking for delivery failure and checks for reminder send by activity monitor. The delivery failure checks for an un-delivered contribution from a member which is returned back and not delivered to the group editing organiser. Then the module tries to find the un-delivered contribution and, when it is found, re-sends it to the group editing organiser. If a reminder of an overdue contribution is received from the activity monitor, the activity responder displays a message, and the member can see the reminder while working with the Newsletter.

Option	Service
A -->	To send edit request to common store
B -->	To edit requested contribution
C -->	To suggest revision to contributions by other members
D -->	To show differences between two contributions
E -->	To create first draft
F -->	To edit earlier drafts (when **xmh** facilities available)
Q -->	To Quit

Table 6.3 Menu for Activity Responder.

- **Send Edit Request**

The module displays a list of all versions of all contributions submitted by the originator and edited by group editing organiser for this originator on any one topic and subject. Each contribution is displayed on one line containing information on; contribution number (unique for its own contributions), date of submission, subject and part of the first line of the contribution. The module requires entry of the contribution number to be released from the common store. The request is sent to the activity monitor carrying the contribution number and a built-in message. The request is required to release a copy of the version by the activity monitor (monitoring the activity) at group editing organisers end.

The display format to show the list of different versions can be changed as required by changing the scan format file available in the Mail directory. For example the update time and/or topic may also be included along with date and subject respectively. The display format is depend on the member who can edit the file using any text editor. However a default format file is defined in the Mail directory.

- **Edit Requested Contribution**

This module checks the copy require to update by means of header parameter '*Version: Released for Edit* ' and *Part-of-Newsletter*. If the copy has been released, the module constructs the header for next version with new updating date and time. The body part of the new contribution is opened for next update with previous contents of the contribution. The contribution uses necessary X.400 messaging services (provided in the message handling environment) to deal with the contribution, such as send, delete, refile list or further edit. The send operation carries the new version to the group editing organiser to be dealt by activity monitor. It is to be remembered that all previous versions are available in the common store and no one is deleted.

- **Suggest Revision to Contributions by Other Members**

This module displays a list of all the contributions submitted by other members. A group member enters the contribution number on which s/he wants to make a suggestion. The activity responder constructs the header for the suggesting contribution. The header parameter carries an additional component '*Revision-Suggested-By: address of suggesting member*'. The contribution uses the X.400 message composition tools to enter the suggestion and appended at the end of the contribution. The contribution then can be sent to the group editing organiser which is handled by the activity monitor.

- **Show Differences Between Two Versions**

This module shows the differences between two versions from a member using standard Unix utility to show difference between two files. The module compares the first and second contributions, then the second and third contributions and so on, in a sequential order. The module displays the difference of compared two versions along with the name of originating member. If the originator has only one or none contribution, it does not display any contribution but the name of the group member. This supplementary requirement is not really an editing requirement, but an enhancement to the tool quality.

- **Create First Draft**

After a group member receives notification from group editing organiser the first contribution can be created by the member. The first notification carries all information about subject, topic, name of Newsletter and submission date for the concerned member which are required to generate the header for the contribution. The activity responder acts on the notification by generating header and body part for the first contribution which is edited by the text editor. The completed draft is dealt in the X.400 messaging mode for further action (e.g. re-edit, refile, delete, send). The first draft can be worked till it is ready to be sent to the group editing organiser.

• **Edit Earlier Drafts (on xmh windows only)**

This facility is only available if the host on which a group member has logged in, supports the xmh windows and the work-station is sparc2gx. This operation takes the member into the xmh environment and provides all xmh facilities: show list of contributions, refile, pick, or edit drafts etc. The old drafts are available in the drafts folder within the Mail directory can be updated during this option.

6.4.3.3 Activity Monitor

The activity monitor is a collection of various sub-modules required to meet the monitoring requirement in the message handling environment. The module needs to run in background to update the status of the activity at a regular intervals. Each event for the activity has to pass through the activity monitor sub-system. The monitoring sub-system deals with two types of transactions: the transactions generated by group members and the transactions generated by its own (activity monitoring).

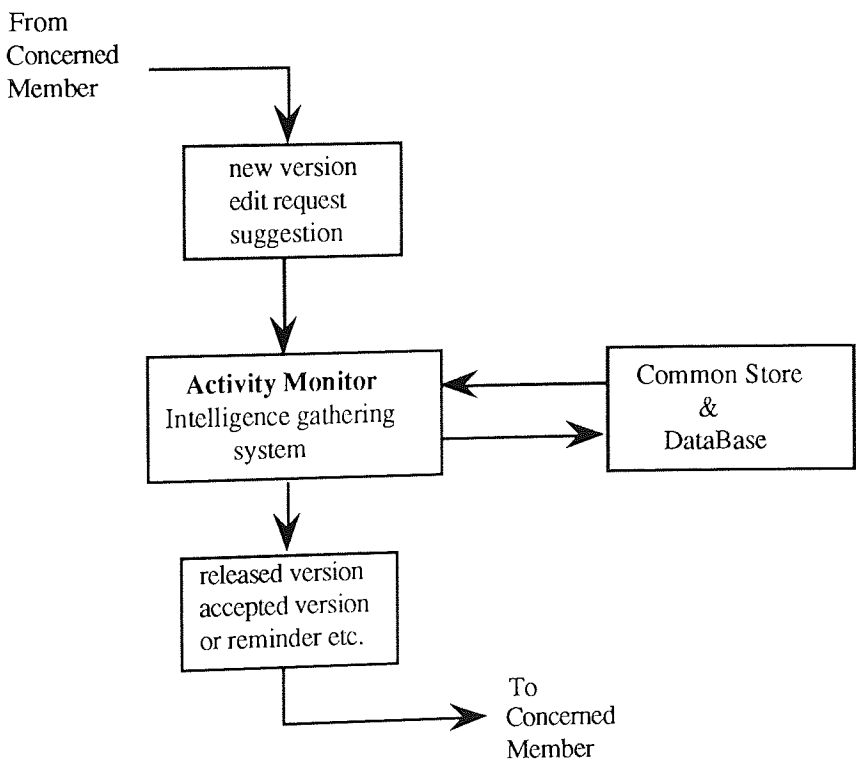


Figure 6.19 Role of Activity Monitor.

Examples of those are, requesting a contribution from common store for next update and an automatic reminder sent by activity monitor whenever it is necessary. Figure 6.19 shows the interaction of activity monitor with member, common store and the rest of the components of the activity.

Most of the features of the activity monitor concern decision making based on the current situation. The activity monitor is also concerned with the collection of information from the editing environment. As the activity monitoring uses a rule-based mechanism for making decisions, it may be termed as a rule based system.

Apart from the monitoring the module does the initial check-up and parameter setting such as incorporating a new contribution, setting-up reminder timing and updating group members list (if any new member is added). The group editing organiser can set up reminder timings as he chooses.

The following are the major monitoring job carried out by the activity monitor.

External transactions:

i) edit request handling,

ii) contribution handling (i.e. linking) and

iii) suggestion handling.

Internal Transactions:

iv) generating reminder and

v) delivery failure checking

When the monitoring system receives an edit request from a member it searches for the required contribution in common store and sends a copy of the contribution to the

member concerned. The released copy carries an additional header component i.e. *'Version: Released for edit'.*

When an updated contribution is received from a contributor by the activity monitor, the contribution is linked by the activity monitor to the rest of the group members as read only, and filed in the folder of the contributor within the common store. In the prototype, if a group member is a networked member, the new contribution is not linked. Instead of linking the contribution, all contributions are temporarily made available to the member, allowing direct access (read only) to the common store. The technique is unsatisfactory and while useful in the prototype it is not suitable for full implementation.

When a suggestion is received from a group member, a copy of the suggestion is sent to the originator of the contribution and the suggestion is filed in the request folder in common store. The suggestion facility not been provided to the group editing organiser in the prototype, although it should be provided in a final solution.

The activity monitor checks for automatic reminder generation. The timing for the reminders are set automatically by the activity monitor but the group editing organiser can alter the setting. At present two reminder times are set. One 15 days from the first notification and the second 7 days after from the first reminder. The system does not provide a reminder facility on an 'as needed' basis (i.e. manual).

Delivery failure is taken care by the activity monitor. If the contribution or message related to the Newsletter is reported by the mail system as not delivered to the concerned member for any reason (i.e. is returned back to the originator of the contribution). Then the contribution is located in the common store and re-sent to the recipient.

This chapter has described the analysis of the Newsletter editing activity and the design of the working prototype which is used for functionality testing and user demonstration. The next chapter evaluates the prototype.

Chapter 7

Testing & Evaluation of Newsletter Prototype

7.0 Introduction

Program testing and user evaluation of the prototype are used in formulating the formal recommendations. There are two phases in the testing of the prototype. One phase is the testing of correct operation of the program code. The other phase is the testing of the prototype functionality against the requirements. The testing of the code is described in Appendix 7.

The prototype is not a final implementation. It is developed to test the functionality and for communication with users to get feedback. Therefore comprehensive error handling, validity of data and integration implementation procedures are not given attention. Hence it is necessary for users to follow the operation manual (Appendix 6) step by step if syntax and run time errors are to be avoided. Errors may cause the prototype system to 'hang-up' or give unexpected results.

The outcome of the prototype evaluation and feedback from the users during the testing phase are given significant weight in providing the final recommendations.The recommendation follows the guide lines from AMIGO and CCITT draft recommendations on group communication, are built-on the CCITT group communication draft recommendations together with the AMIGO project.

7.1 Prototype Testing

The testing process must proceed in stages. There are five stages identified in the testing process (Sommerville, 89). These distinct stages are known as unit testing, module testing, sub-system testing, integration testing and acceptance testing. These phases are shown in figure 7.1. A complete description of step by step testing phase is described in Appendix 7.

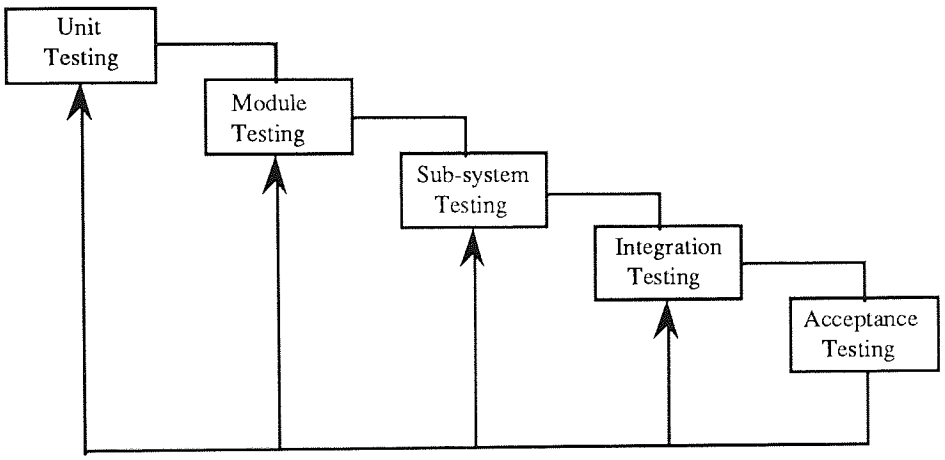


Figure 7.1 Various Testing Phases.

The testing is carried out considering 'ASTON FORTNIGHT' Newsletter, to obtain the requirement specification discussed in section 6.3 of chapter 6. The testing phase also includes demonstration of the editing system as a whole to users to allow them to provide feedback. The system is not primarily concerned with performance, but was used to remove doubts and to improve suitability of the services as editing tools.

7.1.1 Prototype Demonstration

The prototype was demonstrated to Dr. Celia O'Donovan (editor, departmental Newsletter), Ms Susan Jackson and Ms Ghika Duroe (editor and co-editor, Aston Fortnight) and Dr. B. Gay (ex.editor of academic journal).

The demonstrations were pre-arranged with each user and demonstrated in the department of computer science project laboratory. Before the start of the demonstration, a brief verbal description was given about the functionality of the prototype. A printed list of a few pages containing menus, and information related to the menus, was also given to the users just before the start of the demonstration in the project laboratory. To get the comments from the users, sheets were provided which had questions on access, versions, edit functionality, reminder, delivery failure, header, the facility provided in the built-in processes (e.g. information handling). Space was provided for general comments if any. The questions were extracted from the issues raised at the time of interviews. Space of about six or seven lines was provided for each issue.

Each functional operation in the menu were demonstrated with the issuing of an Aston Fortnight Newsletter as example. The demonstration proceeded step by step, beginning from creation of a Newsletter, editing and finally generating the Newsletter. The prototype was operated by a demonstrator. During the operation, each function was explained to the user. At times the users asked questions on the operations. Each demonstration of the prototype took about one and half to two hours.

The facilities for reminders for authors who had not submitted in time and for indicating delivery failure had not been suggested by users in the initial interviews. Following the literature these facilities were included in the prototype, and pointed out to the users during the demonstration.

The comments on the prototype were obtained from the users at later date, written on the sheets provided before the start of the demonstrations. All users considered the prototype functionality would meet their needs, and several improvements were suggested. It was suggested the system should keep on informing members on delivery status, showing date and time of updates while selecting the version for the next update,

that there should be a manual reminder facility, provision of security features to protect privacy and inclusion of a rename function. The key functionality features which were suggested by the users are compiled and listed in the section 7.3.

Apart from the functionality, the users were more concerned about the 'user interface' facilities. It was suggested by the users that the system should be 'user friendly' and facilities provided for such system should be simple to understand. Sometimes users may not be familiar with computer terminology. One user pointed out that such systems are more useful in geographically distributed areas where more people are working together.

7.2 Analysis of Results

The prototype is divided into three sub-systems. Two sub-systems are menu driven and interactive: one for the group editing organiser (the activity generator sub-system) and another for the rest of the members of the group (the activity responder). The third sub-system is a monitoring system (the activity monitor) required to update the status of an activity and to take hidden decisions for the group editing organiser in the background. The Activity generator, monitor and responder sub-systems have been discussed in section 6.4.3 in chapter 6.

During the functionality testing demonstrations were arranged to display the results to potential users for feedback. Rather than discuss the testing in full, only those points that were confusing or needing improvement in regard to users are discussed here. The following paragraphs discuss the functionality of the activity generator, the activity monitor and the activity responder sub-systems in turn, along with limitations of the prototype.

Option	Services
A -->	To Create new Newsletter
B -- >	To Add Group Member
C -- >	To Delete Group Member
D -- >	To Send First Notification to the Member
E -- >	To Process Contribution
F -- >	To History and Statistics
G -- >	To Edit Contribution
Q -- >	To Quit

Table 7.1 Functions in Activity Generator.

7.2.1 Activity Generator

The activity generator acts for the group editing organiser. The table 7.1 shows the functions implemented in the activity generator (placed here for ready reference). The results of the functionality testing of the activity generator are divided into two parts: the **limitation** part and the **user evaluation** part.

There are some shortcoming of the modules in the prototype, which would need improvements when developing a final implementation of the system. The improvements are discussed in the user evaluation phase.

7.2.1.1 Limitations in the Activity Generator

Most of the limitations in the prototype arose because it was decided to avoid complexity during program development. These limitations can be overcome by writing more code or more efficient code, which is not an object here. The easier and quicker program development coding is used to check various functions in the prototype.

The 'Create Newsletter' module restricts creation of more than one Newsletter. This is a prototype limitation which would not be in a full implementation of the system. The reason is that the functionality of the operation is obtained by checking the header component which could be overcome by extending the code.

Another limitation is that the name of the common store is given as 'newsloc' (i.e. derived from news location) in the Mail directory of group editing organiser. The name of the common store should be based on the name of the document (i.e. Newsletter) to create an unique database system for each Newsletter. This can be extended as a sub-directory for each editing activity within the common store, which will form a structured database for such activities. The structure of the Newsletter information is given in figure 6.4.

There is a requirement to create an address or aliases file in the Mail directory of group editing organiser containing all group members for general information. The address file creates a special communication channel between group members. This is created using a text editor. The 'add member' module should include functionality so that it can be created automatically.

7.2.1.2 User Evaluation

Create Newsletter

There is a need to define the group editing organiser separately while creating a Newsletter. The module should consider the Newsletter creating member as group editing organiser (owner of the activity) by default. The status of the group editing organiser should automatically be defined by the function.

In the prototype system a member can contribute only for one topic. For example a member cannot contribute for two different chapters in a book. However, it would be

desirable of one group member could contribute more than one topic in a subject or in different subject. This is a validation problem during the edit request process. The function should be checked for subject and topic for a given member.

Add Member

This module provides dual functionality: it adds a new member in the group and also makes available previous contributions to the new member as read only versions which are in common store. Adding a new member function is straightforward (it updates the information). The functionality to link contributions is achieved by two different ways. The first technique uses the communication link (using the Unix and 'MH' tools) and writes the sequence number of the existing contribution to the new member's 'newsloc' folder in the Mail directory. Therefore only one copy of the contribution is available at common store. A drawback of this technique is that if a communication link fails during the linking process, the rest of the contributions will not be available to the new member. The technique is not recommended for the full implementation of the Newsletter editing system.

A second technique temporarily copies the common store in the 'newsloc' folder in the Mail directory of the new member of the group. This process is carried out when a group member wants to work with the activity, and hence whatever is available in the common store is temporarily copied. The latter technique though expensive in its demands on communication band-width, was used in the testing phase. These two techniques are parallel but whereas the first uses the X.400 functionality, the second requires a direct one to one link (which may be host to host). This is appropriate for prototype demonstration of functionality but not be suggested for full implementation system.

The two techniques described here are only for prototype testing. The final linking mechanism should be taken from the CCITT draft X.gc (Joint ISO/IEC/CCITT, 91) proposed link operations. The operations proposed in X.gc (Joint ISO/IEC/CCITT, 91) are: **linked-to** and **linked-by** which create the relationship to all objects which have the specified link to the named object, and to which the named objects has the specified link respectively (in the form of direct or derived link).

The add member function does not support replacement of a member in the prototype. That is, if a member quits the group, the group editing organiser should be able to transfer the task or contributions of the old member to a new member. A 'rename a member' function to perform this task is an additional requirement for the full implementation.

Delete Member

The prototype module deletes the record of the member from the group member file. This means an un-delete operation cannot be performed for this member. The delete member function should not perform the physical deletion. It should maintain a deleted status character with the record of the deleted member. This will provide flexibility in the operation, and if required the deleted members can be listed or re-included later.

First Notification

This module sends a notification to a member on selective basis. The notification supplies information (on their registration in the group) to a member about their membership in the group. There is no other provision for a member to know that he/she is included in a group. Therefore, the notification should automatically be displayed to the member, whenever a member first logs-in after the arrival of the first notification.

History and Statistics

A limited facility is incorporated to display the history and statistics associated with each Newsletter. The history and statistics aspect contains a wide range of information. It is very difficult to decide what particular information is required at a particular installation. A local installation requirement should be consider as open ended. The history and statistics information may be provided as command line instructions in different break-ups, for example: show requests, show accepted-versions, list group or list suggestions. The command should, for example, be able to display all requests made by a member. It is not necessary to have all functionality as a part of a standard. The history and statistics functionality should have the facility to extend its functionality based on local requirements.

The prototype module does not support the display of information to group members who are not the group editing organiser. It would be better to provide such facilities to all members of the group, so that the status of the activity can be known by all group members. Therefore the group member file should be made available (with restrictions) to all the members to provide information such as history and group information.

Process Contribution

The module combines two functions. Firstly, it accepts from members contributions which the group editing organiser finds acceptable. The 'accept contribution' function does not check for an earlier accepted version. If a version is already accepted, it should give a message saying so. The acceptance of a contribution should stop further updates and any other action taken by an author or reader for this topic. However the group editing organiser should be able to do any modifications, or update the document. This is not implemented in the prototype.

Secondly, 'generate Newsletter' function does not check for defaulters who have not supplied their contribution. During the Newsletter document generating process, the system should check accepted versions from all members. It should provide information to the group editing organiser about the members whose contributions are not accepted and facility to continue the generating process after alerting the GEO to the fact that a contribution has not been accepted. If a Newsletter is generated, the activity should not allow further update by any member of the group and should be closed. However, it should not stop generation of various versions of the Newsletter document. Such access should only be available to the group editing organiser or his/her representative.

Edit Contribution

This module provides facility for the GEO to edit a contribution received from another member. After GEO edits he sends a copy of the revised contribution to the author. In the prototype, the copy contains an additional header parameter 'CC' (carbon copy), which should be avoided in a full implementation. The contribution carries new 'update-on' date and time but the 'CC' header component can create confusion if the original author does not recognise copy (CC) as a modification of his work.

7.2.2 Activity Responder

The activity responder performs two tasks. In the monitoring mode, it checks for reminder and delivery failure during each process.

In another mode, the activity responder provides the editing tools required by the all group members (other than group editing organiser). Table 7.2 shows a list of tools incorporated in the activity responder. The modules, related to this activity sub-system which would need improvements during the implementation implementation of a working system are discussed in the user evaluation phase.

Option	Service
A -->	To send edit request to common store
B -->	To edit requested contribution
C -->	To suggest revision to contributions by other members
D -->	To show differences between two contributions
E -->	To create first draft
F -->	To edit earlier drafts (when **xmh** facilities available)
Q -->	To Quit

Table 7.2 Functions in Activity Responder.

7.2.2.1 Limitations in the Activity Responder

The group editing organiser is able to incorporate a suggestion on behalf of any member of the group. The prototype does not provide the facility for the group editing organiser to make suggestions on another member's contribution. This facility is available to all other members of the group. The facility should be extended to the group editing organiser.

When a contribution is released for update, an edit flag is set by creating a file 'checkedit'. This file is available in the member's Mail directory in the 'newsloc' folder. The checkedit file is also created in the individual folder within the common store. The method is not suitable for a working implementation, because it needs to write a sequence number of the contribution in the checkedit file. The same file is also created over the network in the member's 'newsloc' folder which is risky (may not set a flag) and may not be created if the communication fails during the process.

The edit flag is required to define as a new character (likewise as defined for reply, delete and current) to be incorporated within the contribution. Such characters also required to define for rest of the operations: suggestion request, reminder, new version and current update. The flags for edit request and new version should hold the variable status and should change when the status changes, for example after arrival of a new version, the edit flag should be removed and new version flag should move to the next version.

If an edit flag is incorporated within the contribution, it would avoid duplicate setting of flag, one in 'newsloc' folder in member's directory, another in the common store. This will also reduce communication overheads.

7.2.2.2 User Evaluation

Edit Contribution

A contribution is divided into two parts. The header part and the body part. The header part contains all header components required for an outgoing message and processing. The body part contains the actual contribution written by the group member.

During the edit session the body part of the contribution is opened for update. It does not provides the details of the header component while edit. However, the complete contribution (i.e. header and body) should be displayed during edit session. The reason may not be strong, but it makes clear the subject, topic, to whom it is being sent etc.

When a group member gets a new version for next update the member should be informed at its arrival. So that a member can start next update. No information is supplied in the present prototype.

Suggest Revision

The suggestion contains a contribution carrying an additional header component 'Revision-Suggested-By: *name of the member* '. Apart from an additional header component, a character should be defined for a suggestion which would be able to identify that this contribution contains a suggestion. This character should be incorporated into the contribution in a similar way to the reply character discussed earlier.

Show Differences

The module lists the differences of two consecutive contributions for each member. This does not allow a member to select the contributions that are compared. This facility is made available to all the group members except the group editing organiser in the present prototype.

7.2.3 Activity Monitor

The activity monitor provides a rule-based information handling system to keep an eye on the progress of the Newsletter activity. It is necessary for each event to pass through the activity monitor to update the status. Therefore, the activity monitoring system should check the events related to the Newsletter at regular intervals or as the events arrive.

The reminder generating part should check the necessity for reminders and after certain number of reminders it should inform the group editing organiser. The group editing organiser should be able to send reminders manually.

The system should give indication to the group members for the major events and status of Newsletter to get their attention. If a member is not logged-in, and if there are events, it should inform the member for all events while log-in.

7.2.3.1 Limitations in the Activity Monitor

The activity monitoring system watches each event of the Newsletter activity. When a new contribution arrives from a member the monitoring system processes it. During the processing the new contribution is linked to the rest of the members and refiled in the individual folder for the member. If the communication link fails during the process then the new contribution may not be available to the remaining members. There is a need to extend the functionality of prototype to confirm the contribution, that it is linked to all the members. If the contribution is found not linked to any then it should be linked to that member.

The reminder timings are set (built-in) within the activity monitor. The activity monitor code has to be altered to change the timings. A separate function should be made available to make such timing setting interactive.

7.2.4 Necessary modifications to MHS

Some modifications to MHS are required to enable the 'SAGE' project to meet the required functionality. These are discussed below:

i) New Commands and

ii) Setting up .mhprofile.

Modified new commands are generated from the existing 'MH' tools. For example, to send an edit request, a command (xrequest) is generated from the send command. The component file for the new command is also define as 'requestcomp' (defines modified header components). The new command is also need to declare in the home directory in .mhprofile of the concerned member. The .mhprofile file is to be edited manually.

The Newsletter activity should fall into a group editing application which might contain a wide range of editing activities. This application must be able to access a separate set of tools related to group editing.

The CCITT draft X.gc (Joint ISO/IEC/CCITT, 91) proposes an 'announcement' functionality for asynchronous computer conferencing system. This facility would also be desirable in group editing. There is no provision in this prototype for 'announcement', but it would be desirable in a full implementation. The announcement facility would be useful to find potential contributors to the Newsletter.

7.3 User Comments

As a part of this research, the prototype was used as a testbed for user feedback. The prototype system was demonstrated to assist personal communication with some potential users. The responses from the potential users were very helpful in discovering and clarifying some of the misunderstandings about the services, and gave some new ideas. Some user comments are listed below:

i) Requirement for additional reminders, to be sent manually, as needed.

ii) Requirement to keep members informed of delivery status at all times.

iii) Requirement to provide history and statistics to all members.

iv) Requirement to increase system security: for example a password to entry to the Newsletter activity.

v) User friendliness of the system. These commands should easily be understandable by novice users.

vi) All new events of the activity should be advised to the member when they log-in.

vii) A 'rename' function to replace a member who quits.

The prototype testing and evaluation has allowed testing of functionality of the proposed Newsletter editing model, and also provided user feedback to refine the specifications, and has allowed identification of limitations of the prototype. Two supportive case studies for comparison with the Newsletter are now described in the next chapter.

Chapter 8

Supporting Case Studies

8.0 Introduction

A detailed study has been carried out on editing issues within Newsletter activity. There are other activities whose functionalities are similar to the Newsletter activity. These activities are editing a technical paper and a software development team, which are discussed here to investigate their commonality with the Newsletter activity. The outcome helps in determining common issues in group editing.

8.1 Editing a Technical Paper

The activity consists of the authors and of the operations to carry out editing of a technical paper. The working domain of the paper may span several countries for example, or different, possibly geographically separate, departments of an organisation. The limitation of the domain depends on the network connectivity and the locations of the authors. One of the authors has to act as the group editing organiser having responsibility for completion of the paper. The access rights in this case may be equal for all members. However if a working domain contains a sub-group then the access rights would not be the same at second level of the group.

8.2 Technical Paper Activity

Technical paper writing by more than two people is a joint editing (writing) activity on a single subject/topic. There are different ways to edit a paper. One approach is to edit a single document. In this case the first contribution is updated successively to generate a

higher version. The first contribution can be updated (written to) by any member of the group. This mode of group editing has only two versions of the paper available at one time the latest version and a back-up copy.

A second approach starts from the first contribution which can be updated (written to) by any member of the group. The subsequent versions are generated by editing previous versions. In this approach if the authors of the group are assigned a separate part in the paper than this becomes a similar to the Newsletter. In such case all authors are allowed to update any contribution on their part. But if the authors are not assigned separate task in the paper then the editing has to be made on the latest version, which is in contradiction to the Newsletter activity. In this case updating only the current version would be an appropriate. However the update would follow the edit request mechanism as in Newsletter activity to avoid redundant editing.

The technical paper editing activity consists of the same three base classes (i.e. entity, item and domain). The Group Editing Organiser (who can be a co-author) acts as a coordinator and is responsible for completion of the activity. All authors have the same level of access (read, modify and write) as the GEO, but the GEO has extra administration powers.

8.3 Structure of Technical Paper Activity

The information model of the technical paper activity consists of internetworking and different operations performed by each member of the group (defined in section 5.5). The Group Editing Organiser has additional functionalities for administration and for processing common store, for example: get comments from referee, generate final shape of the paper, activity completion and other administration operations. These operations as in X.gc (Joint ISO/IEC/CCITT, 91), are modelled as follows:

(i) Member port ---> referee, reader(s) or guest editor,

(ii) Subscriber port ---> author(s) and

(iii) Moderator port ---> group editing organiser.

The member port may be available only for commenting on the paper and not for editing. The hierarchy structure is similar to Figure 6.3 in which second layer (i.e. coordinator(s) and co-editor(s)) would be replaced by authors and the last layer by referee & guest editor. It may be possible that author(s) do not know about the referee/guest editor and may not be able to share contributions with such entities.

8.4 Working on A Paper

The selection of subject or topic for a paper, the contributors and the group editing organiser are decided by mutual agreement between authors. These agreements can be reached over any communication medium. The initial agreement need not be via a computer network, but one can use normal Message Handling services to agree these issues before creation of the activity. Once agreement is reached, the activity can be put into operation.

8.4.1 Creation of Technical Paper Activity

The technical paper editing activity could be created by any user over the network. The originator of the activity would act as a group editing organiser. These steps are similar to the Newsletter activity. The originator should be able to start the activity by taking the following action:

(i) Define activity --- The group editing organiser will have to define the activity (i.e. giving name to the technical paper activity),

(ii) Define members of the group --- Creation of associated list of authors, reader(s) and referee(s) group members,

(iii) Define common store --- Creation of common store, and

(iv) Define history file --- Creation of basic information history about the activity like: date of creation, list of authors, reader(s), referee(s), guest editor(s) and latest submission date if any (this file will contain lesser information compare to news letter activity).

8.4.2 Contribution flow

The status of an activity is maintained by the group editing organiser until it is completed. The group editing organiser may negotiate among the members to decide who starts with the initial contribution. Any member of the team can edit the very first contribution, which is submitted to the author acting as the group editing organiser (GEO), who maintains the common store. Further editing is carried out on a controlled basis by making an edit request, and releasing the latest version and each time requested version is released for updating. This will avoid redundant editing (being a single topic/subject editing) among the members. However, any additional text to the topic may be contributed any time by any member of the group. There should be a provision to have sub-sections within the paper so that, different members can work at the same time, if necessary. The Group Editing Organiser would give the required shape to the final contribution. There should be linking mechanism to link it to rest of the members of the group whose master copy is in the common store. Figure 8.1 shows the flow of contribution between an author and the group editing organiser along with the linking process. If each author has a separate part to work than it operates in the similar manner as Newsletter.

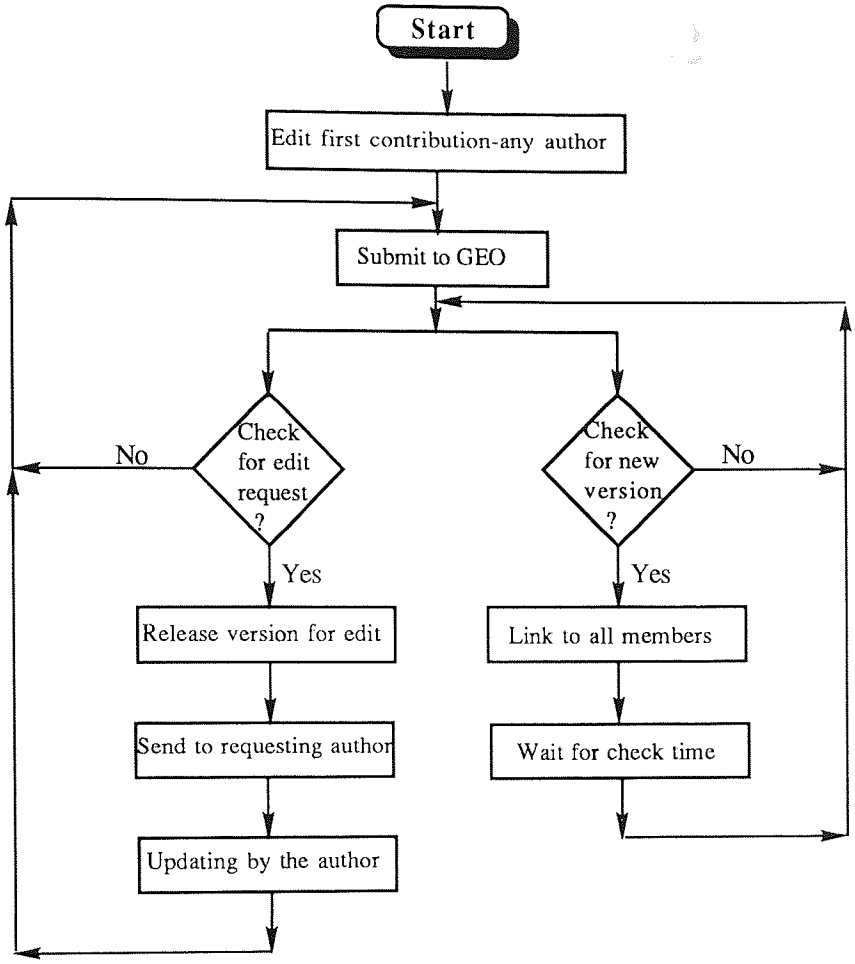


Figure 8.1 Contribution flow between GEO and Authors.

The release of a requested contribution and linking of new updated version should be part of the information handling supported by a service agent. Any reader member should be able to submit a suggestion, which is kept in the common store. The group editing organiser may get comments from the referee/guest-editor. There should be a provision to hide such commented contributions from other members.

Figure 8.2 shows a contribution flow carrying a suggestion by a reader member and figure 8.3 shows the contribution flow between group editing organiser and referee editor in the group.

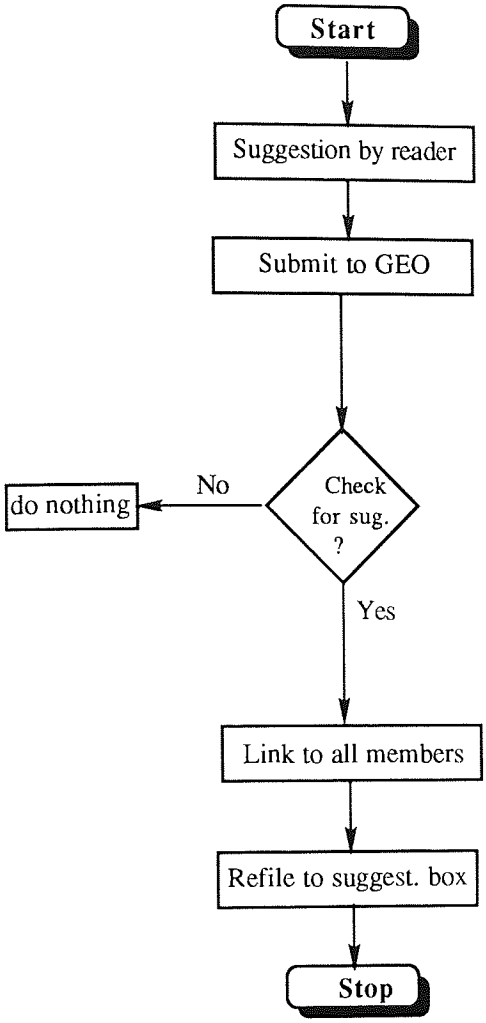


Figure 8.2 Flow of Suggestions.

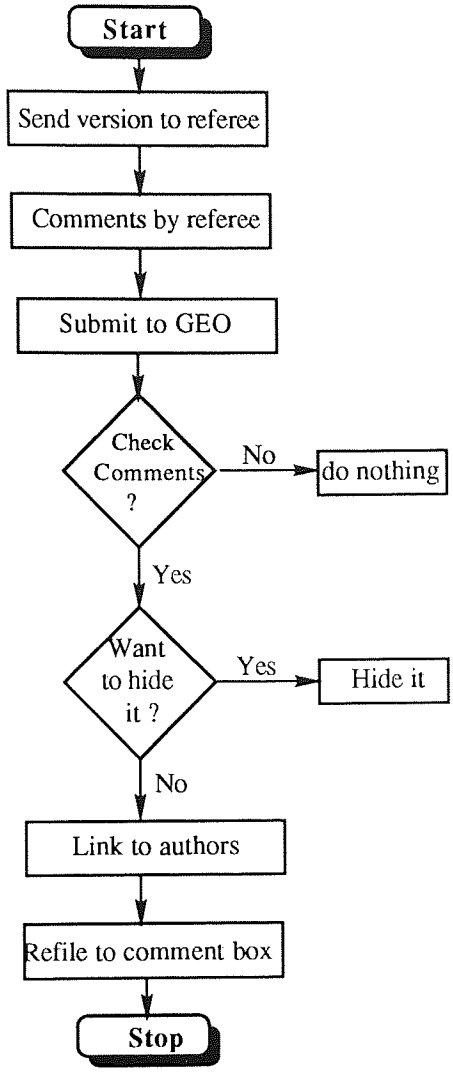


Figure 8.3 Flow of Comments.

8.4.3 Technical Paper Header

The header of a technical paper can be defined using IPMS header extension fields facilities. The component are similar to the Newsletter activity. The group editing header for technical paper would comprise the following parameters which are similar as Newsletter but given separate headings (e.g. paper reference and Authors):

(i) Paper reference: 'name of the paper',

(ii) Authors: 'list of authors',

(iii) Subject: 'subject for paper',

(vi) Topic: (optional in case of sections),

(v) Author: 'name of current updating member',

(vi) Latest submission date: (if any (optional)),

(vii) Suggestion By: 'name of reader/referee' (optional),

(viii) Version: 'Released for edit' (optional),

(ix) Version: 'Accepted' (optional) and

(x) Updated-On: 'date and time' (other then first contribution).

8.4.4 Technical Paper Editing

Any author can start the very first contribution. When this contribution is submitted, it is available in the common store. Since all authors are working on a same document (i.e. paper) on one topic, they are allowed to read or modify (i.e. update) the document. This update has to be controlled by making an edit request, so that only one author can update at one time. If the paper is divided into sub-sections and each author is contributing separately, in such case all authors are allowed to update at the same time but on their own part of the document. However they should be able to read the contributions of other members and the activity is operated in the similar manner as Newsletter.

If comments are required from a referee, the group editing organiser should be able to send such contributions to the referee. A provision for not showing commented contributions to rest of the members or some of the members should be made particularly in this case.

8.4.5 Other Facilities

If the activity is not closed, the retrieval of the document should show the last version of the paper. The group editing organiser can submit the accepted version for publication. A rule-based system would be able to check non-delivery notification and

reminder facilities. In case of delivery failure, the system should be able to resend the un-delivered contribution. If an author fails to submit their contribution within a specific time, an automatic reminder can be sent to the author concerned. The reminder may also be sent in case of a pending edit request. These features are similar to the Newsletter activity.

8.5 Distributed Software Development Team

A majority of software engineering work is carried out in teams which vary in size. In a study undertaken by IBM it is found that 50% of a typical programmer's time is spent interacting with other team members and 30% working alone (Sommerville, 89). The interaction between members plays an important role in the overall performance of the group. A group whose members recognise each other's technical skill and deficiencies and are mutually supportive can achieve better results.

The activity needs comparatively more interaction between the group members. The activity also requires activity specifications in more generalised way, and should be made available on a need basis to each member. The activity has to follow the software development procedure (Sommerville, 89), where testing and integration phase, are difficult in such an environment. Each member has to produce working code as a module before integration.

8.5.1 Software Development Activity

The software development by a group of people is a critical activity in terms of its integration as a package in a distributed environment. Such a package carries pre-defined specifications to produce specified results. Each member of the group is assigned to develop a part of the package in the form of subroutines within the scope of the package. These subroutines are limited to specified input and output parameters. In such cases the activity should define all basic needs of the package like: global

parameters, input, output, data dictionary, system design structure etc. It may possible that the sequence of contributions from each member within the integrated package be defined at early stage. It is necessary for members to test their contribution before submission with the help of simulated data input. However, a member can use other member's contributions from the common store if available. A tested contribution which consists of header and body part is submitted to group editing organiser. These contributions are available in a general folder and the body part of the contributions (i.e. coding only) in the coding folder within common store. The final shape of the coding can be generated by the group editing organiser. The integrated coding contains the chain of body part of contributions in the shape of the package. This considers the highest version of each member. The common store consists of the folders like: general, coding, reminder and modify. The Structure of common store is shown in Figure 8.4.

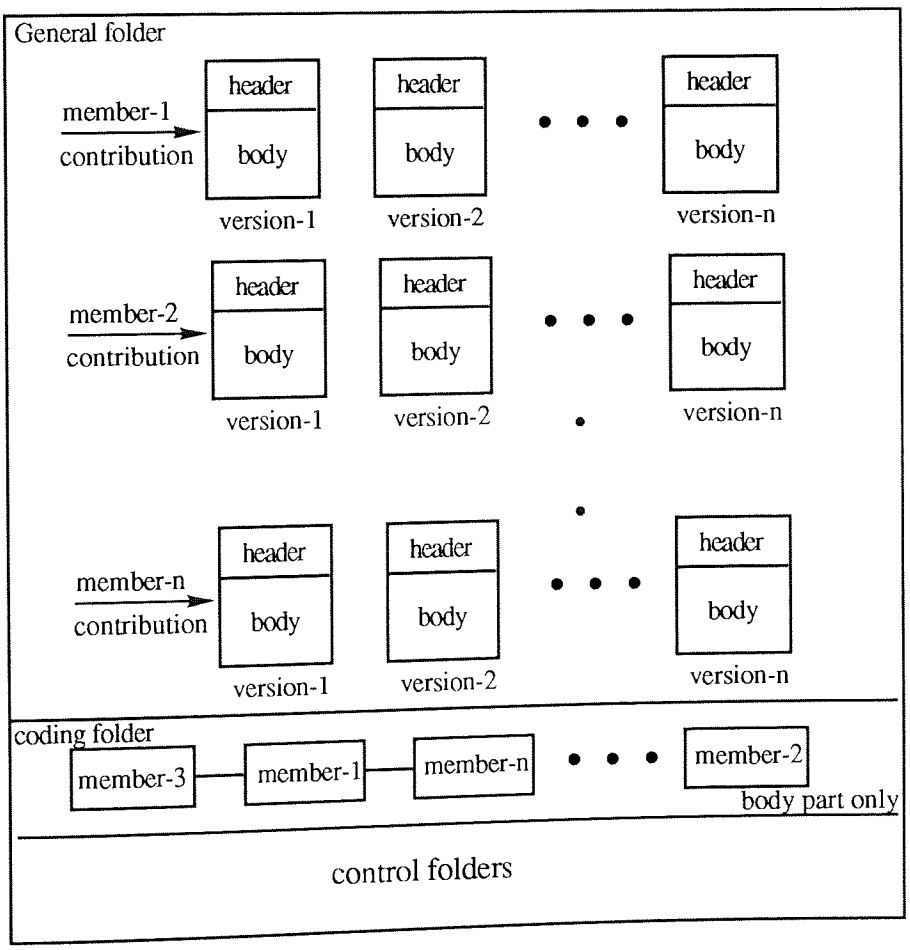


Figure 8.4 Common Store Structure.

Since each member is assigned a pre-defined task within the software package they are usually responsible for completing the task. However, GEO may issue reminders, ask for modifications and assign rights to another member for integration, testing and implementation of the package. The contributions from other member are linked to each member within the group. One can use the contribution for testing purpose. They should not be able to modify the coding of another member. In this activity the control of the group editing organiser is very weak . However, s/he is responsible for administration and completion of the activity testing, integration and implementation.

8.5.2 Software Development Team Activity Working

Like other group activities software development team also comprises the entities, roles and functions in terms of group communication system in a specified domain. The team may consist of members who have the equal status (programmers). The activity may also have experts of the software system who may act as referees at a later stage. The role of group editing organiser (GEO) is like a team manager or co-ordinator. The act of GEO depend on skill (e.g. in case of rejection of code they may write new code). The functional requirement is to be met by the operations and their underlying functionalities. In this case a member has to submit a tested contribution for an integration. The group editing organiser may nominate another member with additional access support for testing and integration of the package. The domain of the activity could be an organisation or a set of organisations.

As described, software development by a team is a critical activity and has its own importance. The activity, compared to others needs frequent interaction between members of the group. It may be necessary to provide the separate channel for interaction between members. This communication channel may also be used for transfer of information other than the contribution which should be considered as a part

of the activity. It would also be necessary that the member should send a copy of tested code to the member who is in need of that code (e.g. sub-routine) as input.

The activity needs to be defined (section 6.2.2.1) by four steps. There would be a necessity to provide the system (software) specification and related information such as global parameters and data dictionary which would normally be accessible by the members. This information would be maintained along with the historical information.

The header components of the activity are similar to those listed in section 6.2.2.4 with a modification. The component '**Paper Reference**' may be given a name '**Part of Package**'

Initially the activity looks like an independent activity. This needs relatively more co-operation when it proceeds towards completion. The operation edit request has weak role in this activity. An author (i.e. programmer) can always submit and improved version of their code before testing and integration.

8.5.3 Identification of Commonality

A comparative summary of the cases for which a study has been carried out is shown in table 8.1. The comparative study showed many similarities in the functionality between the group editing activities.

Functionality Features	Newsletter	Technical Paper	Software Team
Group Editing Environment	Same	Same	Same
Group Editing Organiser Role	Strong	Less Strong	Co-ordination
Access Rights	Based on Status of the member	Based on Status of the member	Based on Status of the member
Common Storage	Yes	Yes	Yes
Multi Versions	Yes	Yes	Yes
Contribution Flow Mechanism	Similar	Similar	Similar
Notification Requirement	Similar	Similar	Similar
Reminder Requirement	Similar	Similar	Similar
Delivery Failure Check	Similar	Similar	Similar
Communication Requirement	Based on Status of the member	Based on Status of the member	Based on Status of the member
Header Component	Similar	Similar	Similar
Create Functionality	Similar	Similar	Similar
Interaction Between Members	very Low	Low	High
Activity Information Interaction	Low	very Low	High

Table 8.1 Comparative Summary between Editing Activities.

For the functionalities that differ, by suitable formulation of their operations and services the functionality can be made sufficiently flexible to accommodate the differences. For example the access rights of GEO would be based on the nature of the activity (discussed in section 5.3), and would be different. Similarly, a header component requires the name of Newsletter or Technical paper or Software package, which would be met by a header component: 'name of document'. This name should be represented by ASN.1 (abstract service notation one) object identifier.

Chapter 9

Proposed Model for Group Editing Activities

9.0 Introduction

The group editing problem is described in section 5.5 on various aspects of editing issues in a group environment. The solution of editing issues in an OSI network environment has been summarised to determine group editing services and tools for group editing activities. This chapter describes a common model for group editing activities considering the results of all three case studies (a newsletter, a technical paper and a software development team), prototype and user comments. Issues on group editing services and tools will be discussed later.

9.1 Group Editing Environment

As discussed in section 5.5.1, an editing environment can be modelled as a functional object. These objects may be identified as contributions, group editing system, and group members.

The group communication services in terms of basic services, advance services and rule-based information handling services (discussed in section 4.3) would be met by a supportive application group agent. The group editing environment consisting of all its components is shown in fig 9.1, within the supportive application group in relation to figure 4.5.

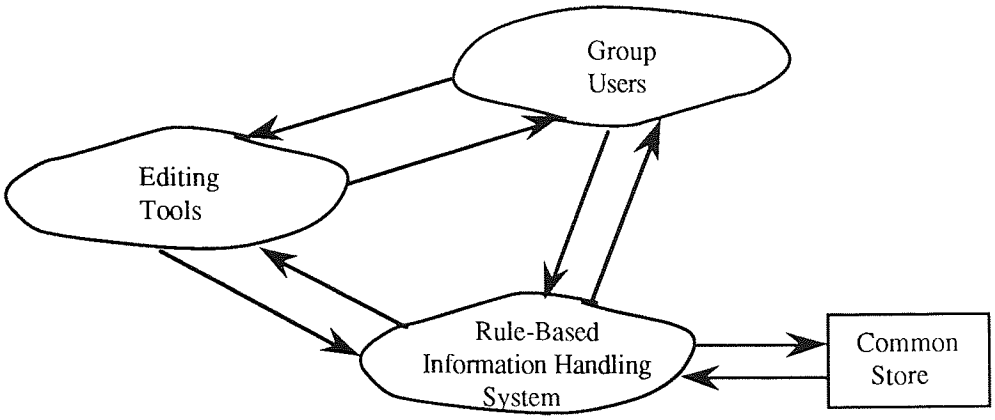


Figure 9.1 Group Editing Environment.

9.2 Access Control

The access control module grants permission to determine which user can perform which functional operations. There are two main reasons behind it. There is a requirement of privacy within the environment where information is shared between users and a requirement of integrity of information. This is the issue for the group editing organiser who can decide the permissions to access these operations (e.g. read and create). The following are the operations compiled from CCITT draft X.gc (Joint ISO/IEC/CCITT, 91) AMIGO report (Smith et.al., 89) and derived from this research to be provided as a group editing tools:

i) Create an activity with distinguished name and attributes.

ii) Add a member with distinguished name and attributes.

iii) Delete a member with distinguished name and attributes

iv) Delete contribution with distinguished name and attributes

v) Send notification with distinguished name

vi) Update contribution with distinguished name and attributes

vii) Accept contribution with distinguished name and attributes

viii) Generate Newsletter with attributes

ix) Show history with distinguished name and attributes

x) Show statistics with distinguished name and attributes

xi) Send edit request with attributes

xii) Suggest revision with attributes

xiii) Show difference of two contribution

xiv) Send reminder with distinguished name

xv) Rename with distinguished name and attributes

xvi) Read contribution with distinguished name and attributes

xvii) List group member with distinguished name

Hidden operations are required in a monitoring system (discussed earlier). The members of the group would invoke these operations by means of three ports which are discussed in AMIGO report.

i) Administration port, which defines operations for administering the activity.

ii) Retrieve port, which defines operations required for retrieval of information objects.

iii) Store port, which defines operations for manipulating information.

The modelling of these ports in a group editing environment is shown in figure 9.2. The figure is extended from AMIGO report (Smith et.al., 89) in which all operations are required to pass through a rule-based information handling system.

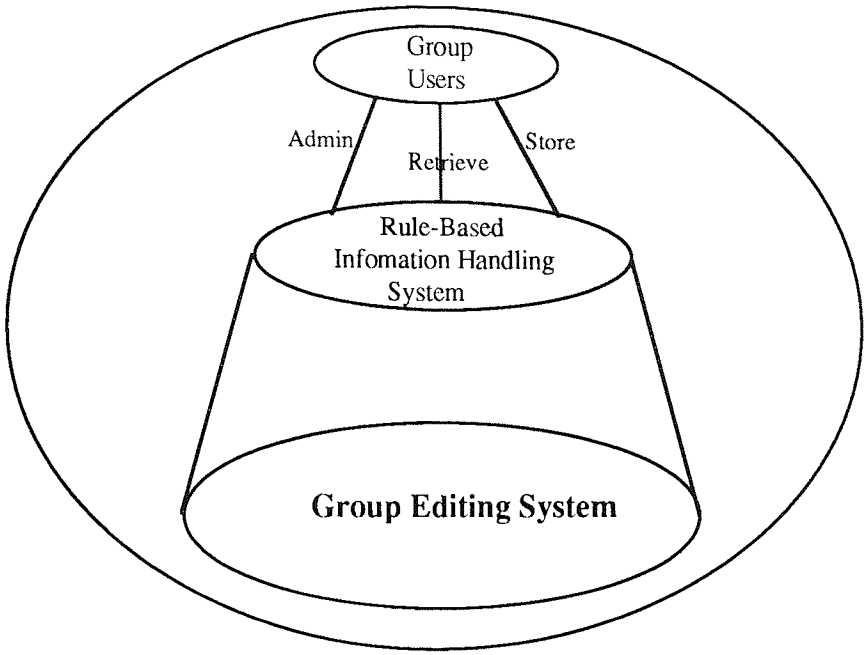


Figure 9.2 Group Editing Environment.

9.3 Storage Facilities

As discussed in the analysis for the case studies, a common work place is required to store the contributions during the editing process by the group of people. Each member has the access rights corresponding to their status in the group. Different storage structures are considered in editing newsletter, writing a technical paper and software development activities. Since each information object is considered as a basic object and separate node, it contains all the information required for processing. Therefore, there is no need to define individual working folders in the common store. A single folder, which can be a common store itself, meets the requirement of a group of people. The approach is different in the Newsletter activity (Ch 5.5.4).

Each event related to the group activity has to pass through a rule-based information handling system. If such rules are incorporated with the newsletter box, it may be called an active 'newsletterbox' within the mail directory. The 'newsletterbox' may act as active mailbox on incoming and outgoing messages in regard to editing activities. This

'newsletterbox' may consist of advanced group communication support services to handle the information to and from the common store.

The AMIGO MHS+ report on use of directory services for group communication (Weiss, 89) discusses the existence of an archive agent to deal with the services related to the common store. This means the rule-based information part related to common storage handling may be built-in as archive agent. Services such as request handling, linking of new arrivals, suggestion handling and history and statistics could be integrated as a part of the services of an **archive agent**. The AMIGO MHS+ report also suggests defining a document identification (document-id) as the directory object class which can be used to request the document.

A mechanism for linking contributions in common store to all the members (as read only) is necessary to avoid multiple copies of a contribution. This reduces the disk space overhead for large activities. It could be part of advanced group communication services.

The 'SAGE' project strongly supports an entity **archive agent** (to provide store and retrieve services) with its extended functionality. As discussed in section 4.3 the services of supportive application group agent (SAGA) would contains basic services, advanced services and information handling services. The services of an archive agent would be used as a part of store and retrieval services within the SAGA. The services of supportive application group agent provide the group communication services support for the common store. Figure 9.3 shows the proposed relationship structure between support services and common store considering the monitoring part as an archive agent. A group member has to use the services provided by the message handling system, the directory and the services forming group editing tools to complete an editing activity.

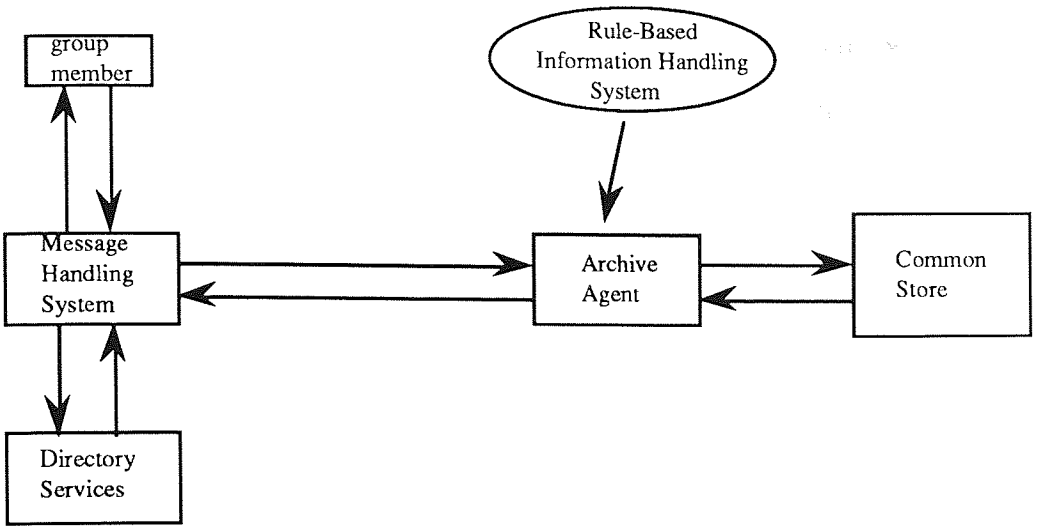


Figure 9.3 Common Store Relationship with Other Services.

The AMIGO report describes the requirement of the internal data information model. A similar approach is used for the structure of the common store, which must be able to meet the following requirement (Nunez, 89):

i) it must define basic communication objects.

ii) must represents the information needed for handling distribution of information.

iii) it must define elements and operations to support distributed information access.

iv) it must support the multi-user characteristic of the system along with access control and privacy.

v) it must support a namespace of globally unique names.

9.4 Version Handling

A contribution is made up of a header and a body part. The header of the contribution contains a component 'Updated-On'. There is a direct mapping between the fields and the attributes of a contribution. That is, the 'Updated-On' field contains the date and time of submission of the contribution. This date and time is derived from the system during submission, which is unique for each contribution. The posting time of the contribution is used to distinguish various versions. The timing would also be helpful

for controlling versions. To identify the last version a character should be incorporated within the contribution which would indicate the last version. Figure 9.4 shows various versions in the common store submitted by a member.

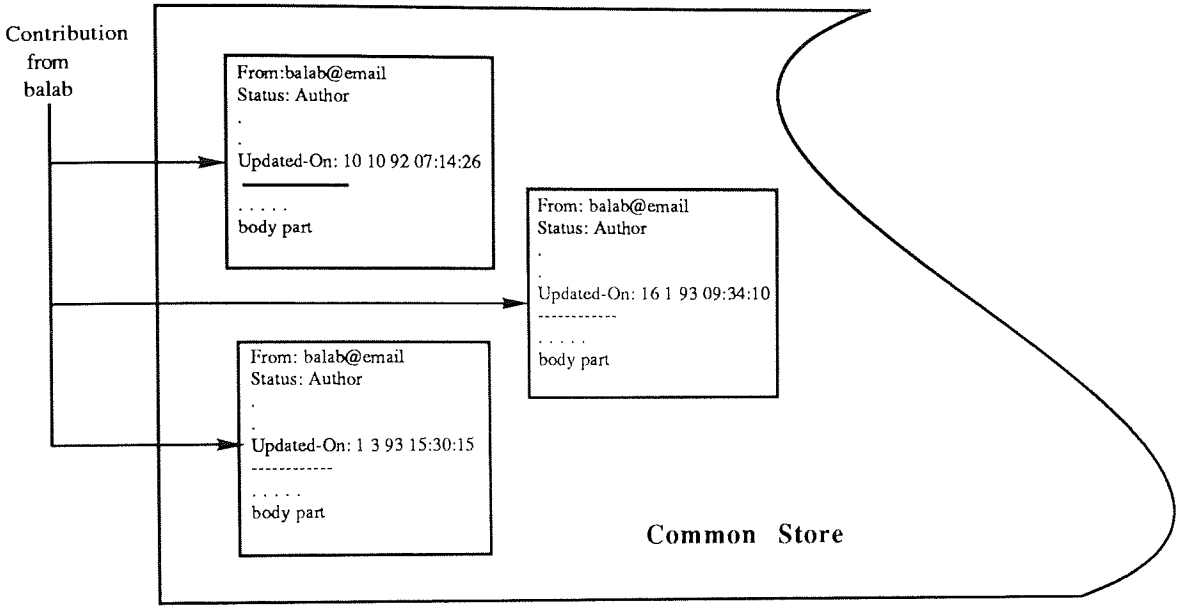


Figure 9.4 Different Versions by a Member in Common Store.

Apart from this MH tools can also be used to check the last previous and first version of a contribution. A list of such useful commands are placed in Appendix 6 with operation manual.

Chapter 10

Recommendation for Group Editing Services

10.0 Introduction

Based on the study made on group editing activities which are discussed in chapters 4 through 9, the group editing services are recommended. The recommendations are proposed in the style of CCITT and are comprised of Abstract Service Definitions mapping onto the message handling system, and functionality extensions to the X.420 protocol.

The recommendations style are discussed in the light of global naming structure and represented by ASN.1 (abstract service notation one) object identifiers (Joint ISO/IEC/CCITT, 91). The abstract service definitions are given for the operations which are additional to the CCITT draft X.gc (Joint ISO/IEC/CCITT, 91) for group editing activities. The mapping and X.420 protocol functionality extension is proposed in terms of component files (header components) readily available for editing operations.

10.1 Group Editing Services

The group communication services for editing application (within the supportive group) are projected in terms of the group editing services tools as basic services, advanced services and rule based information handling system. The editing services are the only part directly related to the users of the system. The rule based information handling system is the underlying functionality required by the editing application group as a whole to manipulate internal and external transactions.

10.1.1 Editing Services and Tools

The 'SAGE' project view on a functionality of any group activity is that it can be divided into two parts: the **tools part** and the **services part**. The first part comprises the operations directly accessed by the users of the group. The second part is the underlying functionality required to meet such operations. For example, an 'edit request' is a functional operation which would present a contribution to a member for next update. When a member sends an edit request the underlying functionality involved in the operation is: the request is sent to the common store, the required contribution (version) is searched, the availability for next update is checked and if found to be up-datable then the contribution is retrieved over the network for next update. The functionality involved with the operation is depend on the complexity of the operation.

The 'SAGE' project defines the operations required by different roles in the activity as a tool set. The underlying functionality of these tools is to be provided by an application service agent. The underlying functionality provided by the service agent is defined as the services for an activity. Therefore a group communication system would provide these services and tools to meet the requirement of the activity. The services could include a rule-based information handling requirement.

The support requirement for a group editing environment is split into two parts: the operational part of the functions and the rule based information handling part. The operational part of the functions is discussed in this section. The group editing requirement is worked out in chapter 9, while discussing the group editing model. The set of abstract operations which access and manipulate group communication editing objects are defined in turn. The conceptual group communication operations are specified in terms of outline functions in the form (Benford et.al, 90):

function name (parameters) --> results

The operations are specified using OSI Abstract Service Definition conventions. The abstract operations should return status information indicating their success or failure as a part of the operation results. The operations must be given distinguished names to identify objects.

10.1.1.1 Naming

The naming of objects is a critical for large distributed systems (Benford et.al., 92). All objects are accessed by names provided them. Therefore each object must have at least one globally unique name. Group Communication Distinguished Names (DNs) provide the handle by which all objects accessed (Benford & Palme, 93). An object may be identified by more then one name (e.g. aliases). It should be possible to locate the object by a given name in a globally distributed system. Such examples in this project are: name of a group editing activity, group editing user, name of common store. In the description **DN** stands-for **Distinguished Name** and **group of DN** is a **clusterDN**.

The **Distinguished naming** mechanism should be defined in same way as in CCITT draft X.gc proposal consisting of an ordered sequence of attributes type/value pairs. The name should be divided into two components; the domain part and the local part. The domain part indicates the naming authority for the object and the local part consists of a subset of the objects attributes which must be unique within the domain. A domain is itself an object with a name where each name part is an ordered sequence of attribute type/value pairs. For example:

organisation=X@organisational unit=Y@topic=editing activities,

where name of a cluster may represent an editing group activity which would be contained in its naming organisation domain, organisation = X@ organisational unit = Y (**domain part**), and the local part of the name would be topic = editing activities (**local part**), the symbol @ is used to separate name parts and represents an attribute type/value pair in the form type = value. The attribute types are globally unique and are represented by ASN.1 objects identifiers.

The internal state of each object is represented by a set of **attributes** (i.e. **attributelist**), each of which has a **type** and a set of **value**. Attribute types are globally unique (Joint ISO/IEC/CCITT, 91) and represented by ASN.1 (Abstract Service Notation one) object identifiers. Attribute values may have complex structures within ASN.1 syntaxes.

Figure 10.1 shows an example of an attribute in an information object (Joint ISO/IEC/CCITT , 91). In the context of the editing activities each header component may be viewed as a single attribute (e.g. topic, subject, updated-on, status of the member and a group member).

Type	Value
IPMessageID	123@John
Importance	High
Recipient	aphjordan@hud.ac.uk balab@email.aston.ac.uk bernard@quipu.aston.ac.uk

Figure 10.1 Attributes in an Information Object.

The external relationship between objects are represented by **links**. A list of such link operations defined in CCITT draft recommendations are placed in Appendix 4. Each link has a type and restricted to the editing group objects. For example:

- Membership links between entities and domains,
- Contribution links between items and domains and
- Update links between items

10.1.2 Abstract Service Definition

This section specifies the Abstract Service Definition of an asynchronous group editing communication system. Asynchronous means that there is no support for 'real-time' communication between group editing users. The system works on store-and-forward manner (as defined in X.400 recommendations). The editing group consists of contributions, topics, subjects, documents and participants. The group editing environment defines certain advanced group communication services required in context of editing a document. It does not specify an implementation specification for such protocols as a whole.

The abstract operations access and manipulate group editing objects. In broad terms the operations support retrieval, storing, administering and modification of editing objects. The access right is depend on the nature of the activity (Benford & Palme, 93), viz: open, closed, restricted or protected. This is an issue to be decided by the owner of the activity i.e. the group editing organiser, that what kind of access to be provided to various members of the group.

The abstract service definition specifies operations which allows groups to work with the editing document. For example: create, update, suggest and read complex shared information structures within domains. The editing group users would interact with the group editing system by invoking abstract operations by means of admin, store and

retrieve ports (discussed in section 9.2 in chapter 9). These operations are informally described below.

Create-Activity (DN, domainDN, attributelist) --> attributelist, status

Creates editing activity with the given Distinguished Name and attribute in the Distinguished domain. The object should be created and the name of the common store is to be determined from its name. The object type identified the class of object. The object can subsequently be removed from its distinguished domain.

Register-GroupMember (DN, clusterDN, attributelist) -> status

Operation which is able to create an editing group for a given activity with the status (for example author, co-ordinator, reader etc.) for Distinguished Name activity within the specified domains. The attribute list is a set of values containing topic, subject and status of the Distinguished Name group member. The operation allows a member be a subscriber within the specific domain. It is to say that this operation registers a user to be a group member and defines a **cluster** of **Distinguished Name** members. The operation creates a relationship between the Distinguished Name activity and the Distinguished Name editing group member for specified set of attributes.

Delete-GroupMember(DN, clusterDN, attributelist) ->attributelist, status

The deletion operation should delete the specified object (group member) from the clusterDN for the given attribute list, and for distinguished name activity. It should remove the object from all domains and clusters in which it exists. The operation should be able to unlink the relationship between specialised objects for a given attribute list. The value of attribute may contain a particular subject and/ or topic. The status of the operation must indicate the success or failure along with the attribute list.

Edit-Notification (DN, clusterDN, attributelist) -> status

This operation should support sending of a first notification of attributes belonging to a single Distinguished Name. The operation locates the named object and sends a notification, and establishes the relationship between the Distinguished Name (i.e. activity) and the clusterDN (group) address. The attribute list can contain values (e.g. latest submission date) by adding the member in the group for a specified object. The status of the operation must indicate success or failure for the notification.

Update-document (DN, clusterDN, attributelist) -> attributelist, status

The operation locates the named object and checks for the availability for next update. If the contribution specified by the attributelist values is free to update, then presented for next update. For example a group member requests access to update a contribution specified in the attribute list (for a topic and subject) for a group edit activity (DN) which was requested to update. If the contribution is released from the common store then it should be presented for next update. The operation should return the status value as attribute list (e.g. can't be released or being released or not authorised etc.) as success or failure . In case of first time edit, the operation should check the notification and return the status accordingly, with the appropriate action.

If a suggestion is considered to be incorporated, the next update for the specific topic should be locked. A built-in functionality should be provided on the request of the originating member or the group editing organiser, for incorporating the suggestion. After submission of the contribution containing the suggestion such locking should be removed.

Edit-Request (DN, attributelist) -> attribute, status

The operation should be able to locate the Distinguished Name (activity name) and specified attribute objects (e.g. contribution value and topic etc.) and retrieve it for the

requesting member from the common store. The edit-request operation should set the edit flag to stop next update on this topic until arrival of new version. The status indicates the success or failure of the operation. A failure should return a message (e.g. contribution under update: wait for new version). The Distinguished Name (activity) should establish the relationship with the contribution and the common store.

Accept-Contribution (DN, clusterDN, attributelist) -> status

The operation accepts the contribution for a document from Distinguished Name group member. The operation is able to set the flag not to be further updated or to entertain request handling for given attribute list objects (e.g. given topic and subject). Success or failure status gives an adequate message: such as contribution already accepted or document has been completed.

Generate-Document (DN, attributelist) -> attributelist, status

The operation generates the Distinguished Name editing document in the order of a given attribute list. The operation should allow generation of more than one format for the document. However, it should not allow any change in the document by any member but the group editing organiser. The operation also allows generation of the document in the sequential attribute list decided by the group editing organiser. The status of the operation should indicate defaulters whose contributions are not accepted and continue if desired.

List-GroupMember (DN) -> clusterDN, attributelist, status

The operation returns the Distinguished Name of the cluster objects and the name of the type of all Distinguished Names who have a relationship with the given Distinguished Name activity. The status should indicate the success or failure of the operation. The attribute list specifies the Distinguished Name members other details, such as his/her status, topic, subject etc. for a given Distinguished Name activity.

List-Statistics (DN) -> clusterDN, attributelist, status

The informative operation returns statistics of all objects (contributions e.g. notification, total contribution, accepted contribution etc.) for a Distinguished Name activity along with the Distinguished Name member. The list of attributes should be treated as local requirement. Such requirement may vary activity to activity.

Edit-Suggestion (DN, clusterDN, attributelist) -> attributelist, status

The operation allows a member to append text in a Distinguished Name activity for a given Distinguished Name member. The attribute contains the contribution object which need suggestion. The operation also sets a suggestion flag within the contribution and carries an additional header component 'Revision-Suggested-By'. Success or failure should also be indicated by the operation.

Show-Difference (DN, clusterDN, attributelist) -> attributelist, status

The operation is required to show differences between two contribution objects (different versions) by a cluster Distinguished Name group member for a given Distinguished Name activity. The status should indicate the existence of the versions, success or failure.

Rename-Member (DN, clusterDN, attributelist)

--> DN,attributelist, status

The rename operation re-assigns the objects (contributions) of Distinguished Name group member who no longer has a relation with a given attribute type in a Distinguished Name activity. This operation replaces the Distinguished Name group member with the other Distinguished Name group member by changing its name. The status should indicate the success or failure of the operation.

Edit-Reminder (DN, clusterDN, attribute) -> status

This operation issues manual follow-up reminder to cluster Distinguished Name group member for given attribute type Distinguished Name activity. The success or failure status is also returned by the operation.

Delete-Contribution (DN, clusterDN, attributelist) --> attributelist, status

The operation deletes an object contribution submitted by a Distinguished Name group member on a given attribute type Distinguished Name activity. The attribute list should conform the status of an object. The operation should also indicate the success or failure.

Read-Contribution (DN, clusterDN, attributelist) --> attributelist, status

The operation retrieves information about a single object (contribution) to read for a Distinguished Name group member on a given attribute type Distinguished Name activity.

Show-History (DN) --> attributelist, status

The operation returns a list of individual objects related to the Distinguished Name activity. For example it should include the name of the activity, completion date, name of GEO.

The operations such as create-activity and delete-contribution have also been defined in X.gc (Joint ISO/IEC/CCITT, 91). These operation would possibly be modelled on create and delete operations as also proposed for editing activities.

Other operations which have already been specified in CCITT X.gc draft (Joint ISO/IEC/CCITT, 91) as basic operations or 'GRACE' project (Benforrd et.al., 90) for

group communication services are not mentioned here. A list of such operations (in brief) is placed in Appendix 4.

10.1.3 Group Editing Information Handling system

A brief discussion is made in section 9.1 while discussing group editing model of the rule-based information handling system. The group communication working document X.gc (Joint ISO/IEC/CCITT, 91) specifies an entity i.e. group communication service agent (GCSA) for basic group communication system services. The idea of having an archive agent proposed by Weiss (Weiss, 89) in AMIGO report is considered to be a key issue in rule-based handling information. The archive agent would provide the services for retrieve and store information to and from the database or common store.

As described in section 4.3 there would be a user service agent for each application group, which would meet the functional requirement of the activity. The 'SAGE' project proposes an entity **supportive application group agent** (SAGA). The entity supportive application group agent would be a combination of rule-based information handling services, basic group communication services and the rest of the advanced support required for a group editing activity. The services of a group communication service agent should be integrated as a single user agent for such application group. Figure 10.2 shows a merger of such entities in a single unit as a supportive application group agent.

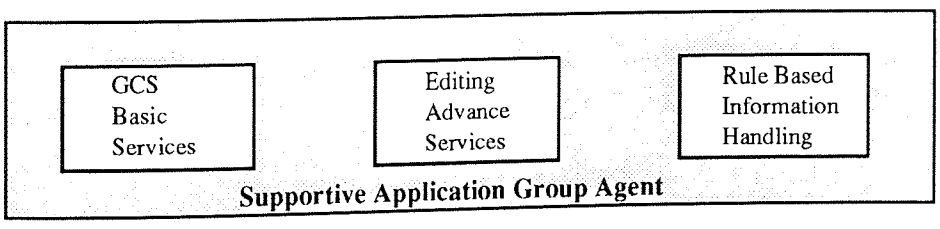


Figure 10.2 The Supportive Application Group Agent.

The services provided by the supportive application group agent (SAGA) would contain rule based decision making support for the editing activities. If the agent is mapped onto the present prototype, it is to say that the SAGA should provide all services incorporated in the activity monitor, for example: edit request handling, linking relationship between contributions and group members, checking delivery failure, automatic reminder generating, selection of various header component on an specific event. The agent should also be able to take decisions for storing, retrieval or administering the editing application group.

The underlying functionality of SAGA is a part of monitoring system. Every event of the group editing system should pass through the SAGA. The underlying functions are able to generate internal and external transactions. The functionality would be meant to support a store and forward system i.e. X.400.

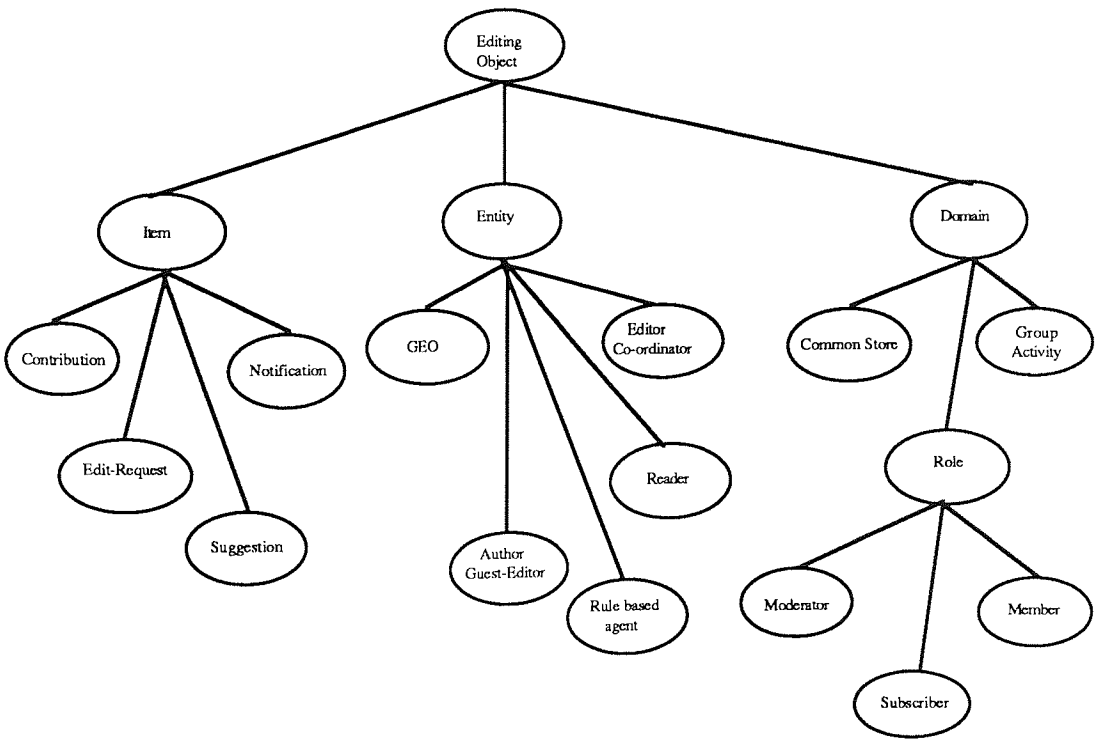


Figure 10.3 Editing Activity object classes derived from GCS object classes.

10.2 Mapping the Services Onto MHS

An editing activity consists of a number of objects and roles. The objects can be identified as items (i.e. contributions), editing groups and sub-groups and a domain. An editing group which is an object contains a number of items on an editing activity. The items are submitted to the group editing organiser, who can accept, ask for update, suggest or edit the item. Figure 10.3 derives the group editing activity objects classes from group communication system (GCS) object classes. In addition figure 10 shows Entity Relation Diagram for group editing activities (Avison, 85).

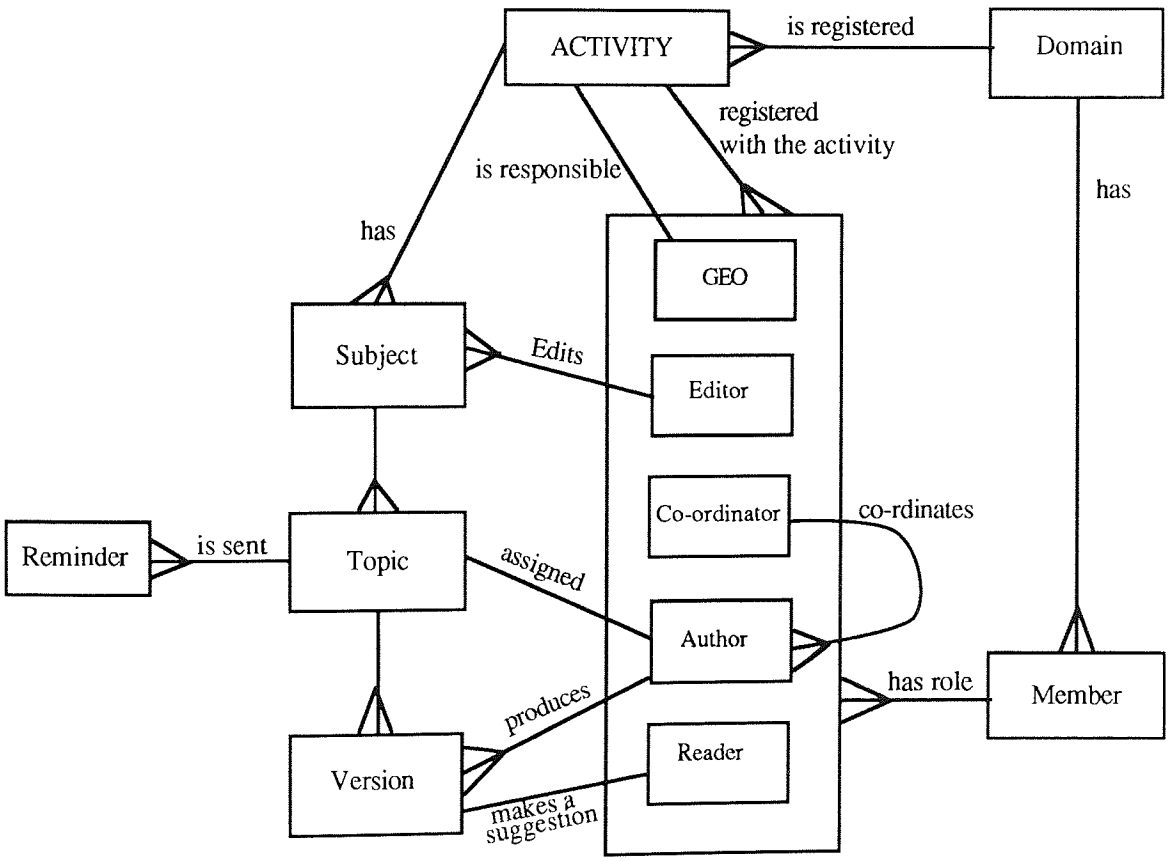


Figure 10.4 Data Model for Group Editing Activities.

The domain defines the group, within which its items or contributions are visible. Typically, this should default to the area domain where the group is named. If an editing

group has a sub-group, a similar kind of mechanism should be provided to meet the requirement. It should follow the X.400 addressing format (area domain, organisational domain).

Each user entity in the group performs a role within the group based on a defined set of operations. An editing group consists of the following roles:

- Group Editing Organiser,
- Author,
- Editor,
- Co-ordinator,
- Group Editing Organiser,
- Guest Editor and
- Reader.

Group editing organiser is responsible for initialising an activity. The group editing organiser is also responsible for the following operations:

Create-Activity	creates a new activity for group edit
Define-Group	registers group members with the activity
Delete-GroupMember	removes a member from the group
Edit-Notification	sends an initial notification to a group member
Accept-Contribution	accepts a contribution as a part of edit document
Generate-Document	integrates all the accepted contributions in a single document
Rename-Member	re-assigns the contributions of an old member to a new member
Delete-Contribution	removes un-wanted contributions from the common store.

The role of editor and co-ordinator may also be viewed as area or organisation manager/ administrator. The role of an **Editor/Co-ordinator** will be a sub-set of those of the group editing organiser but limited to the scope of the area or organisational domain. Such operations are listed below :

Define-Sub-Group	registers a member within the sub-group
Delete-Sub-Gr-Member	removes a member from the sub-group
Edit-Notification	sends an initial notification to a group member
Rename-Member	re-assigns the contributions within sub-group
Delete-Contribution	removes un-wanted contributions from the common store
Edit-Reminder	sends a follow-up note to a specific member for a contribution.

The role of an **Author:**

Update-Contribution	updates a released contributions or it could be a first edit
Edit-Suggestion	appends text to a contributions as a suggestion
List-groupMember	returns the list of group member
List-Statistics	lists various contributions from all members
Show-Difference	shows differences between two contribution from a member
Show-History	shows activities history viz: date of completion, GEO etc.
Read-Contribution	reads a contributions from a specific member.

The guest editor is able to comment on a contribution. If guest editor is asked for such comments with the help of 'Edit-Suggestion' operation is used for the comments. As such a guest editor is a silent member of the group. The role of reader is also similar to the guest editor, but a reader member would not be asked for comments. However a

reader member is able to suggest. The capabilities of an **Guest-Editor and Reader** are:

Edit-Suggestion	appends a text to a contribution as comment/suggestion
Read-Contribution	reads a contributions from a specific member.

The information model consists of information and group members as objects. Objects in the group editing environment may be illustrated as below:

Items

contribution
suggestion
edit-request
follow-up call (reminder)
filter (filter keeps a record of which items have been read and currently unread)

Cluster

edit group

Domain

activity domain
role domain
area
organisation

Entity

group editing organiser
editor
co-ordinator
author
reader
any process agent

10.2.1 Services of X.400/X.420

An information object represents a single logical item created by a member of the group. Each information object in the editing group has a header containing information identifying the object and a body containing text. The header part consists of a number of fields including author, part-of-document, subject, topic, updated-on. The information objects governs X.400 messaging format. That is to say an information object is made up of an envelope and the content (discussed in section 3.3 in chapter 3). The envelope carries information that is used by the message transfer system (MTS) when transferring the message within the MTS.

The header requirement (i.e. within the envelope) in the information objects are met by the use of X.420 extension fields facilities. The extension field facilities exist within the 'P22' protocol used for InterPersonal Messaging System (IPMS). The facilities of 'P22' protocol are extended to meet the requirement (including part-of-document, updated-on, latest-submission-date and status-of-member) for a group editing environment.

The heading part of IPMS i.e. X.420 recommendations (discussed in section 3.3 in chapter 3) defines several kind of fields component types such as: IPMIdentifier, Recipient specifier and O/R descriptor (a list can be found in Appendix 2). The IPMIdentifier is used to define various application as ASN.1. At present the '11' is the only ASN.1 application defined for person to person communications. It is proposed to define another **GroupIdentifier** for **supportive application group** which would provide the services required by the editing activities. The identifier should also be able to set-up communications requirement between the group users for such group. The Abstract Service Notation One (ASN.1) definition for the services in terms of the operations proposed in the 'SAGE' project are given in Appendix 8.

10.2.2 Message Handler Services

The services provided by the message handler are in the form of executable commands. For example to compose a message, a compose command is used which requires a component file. A similar kind of approach is used to define the various group editing commands. These commands are described as Abstract Service Definition in the section 10.1.2. The mapping of these functions onto message handler will be described in turn in the following paragraphs.

The commands Edit-Notification, Update-Document, Edit-Request, Edit-Suggestion and Follow-up-Call (reminder) are of the similar nature. These commands can be generated by creating a new command from compose command of the message handler, which would be derived from component files specific to each command. For example 'Edit-Notification would be using a component file header components for these component files would be containing specific header parameters described below. The requirement of additional header parameters could be met from the X.420 extension field facilities.

Header parameters for **Edit-Notification component file** 'notecomp'.

To: clusterDN (address of the group member)
From: geo
Status: Author (to whom notification is to be sent)
Subject: attribute (any value of attribute)
Topic: attribute (value of attribute)
Part-of-Document: DN (name of the activity)
Latest-Submission-Date: attribute (value of attribute)

The **Update-Document component file** 'updatecomp' should contain the following header parameters.

To: clusterDN (address of geo)
From: clusterDN (updating member)
Status: Author
Subject: attribute (any value of attribute)
Topic: attribute (value of attribute)
Part-of-Document: DN (name of the activity)
Latest-Submission-Date: attribute (value of attribute)
Updated-On: attribute (value of attribute)

The **Edit-Request component file** 'requestcomp' should contain the following header parameters.

To: clusterDN (address of geo)
From: clusterDN (updating member)
X-Number: attribute (any value of the contribution number to be released for update)
Subject: attribute (any value of attribute)
Topic: attribute (value of attribute)
Part-of-Document: DN (name of the activity)

The header components for **Edit-Suggestion component file** 'suggestcomp' are as below.

To: clusterDN (address of geo)
Revision-Suggested-By: DN (name of the suggesting member)
Status: Author
Subject: attribute (any value of attribute)
Topic: attribute (value of attribute)
Part-of-Document: DN (name of the activity)

The header components for **Edit-Reminder (Follow-up-Call) component file** 'remindercomp' are given below.

To: clusterDN (address of the group member)

From: geo
Reminder: clusterDN (to whom reminder is to be sent)
Status: Author (to whom notification is to be sent)
Subject: attribute (any value of attribute)
Topic: attribute (value of attribute)
Part-of-Document: DN (name of the activity)
Latest-Submission-Date: attribute (value of attribute)

The header components for **Create-Document component file** 'createcomp' are given below.

Application-Identifier: attribute (pre-defined value)
Work-Domain: domainDN (common store location in the domain)
Document-Name: DN (name of the activity)
Completion-Date: attribute (value of attribute)
Special-Comments: attributes (value of attribute)

The header components for **Generate-Document component file** 'documentcomp' are given below.

Application-Identifier: attribute (pre-defined value)
Document-Name: DN (name of the activity)
Completion-Date: attribute (value of attribute)
Order-of-Accepted-Versions: attributes (value of attribute)

The functions register-member, delete-member, delete-contribution, accept-contribution and rename-member would be able to return the value of status. For example, in case of delete-member, success status would contain value like 'the "X" member has been removed from the group' or failure status would contain value such as 'can't delete "X" member or "X" member not found' which ever is suitable for the situation.

The operations, list-groupmember, list-statistics, show-difference, read-contribution and show-history would return the value of attributes in case of success, failure status otherwise.

Chapter 11

Conclusion

11.0 Introduction

The 'SAGE' project has sought to investigate the needs of a group of people working co-operatively and communicating in an OSI network environment, and to recommend tools and services to meet these needs. The research has focused specifically on the services and tools required for group editing activities.

The work of the 'SAGE' project is in the application layer of the OSI model, and is particularly concerned with the X.400 (i.e. message handling system) series of recommendations.

The 'SAGE' project has followed recent suggestions (Benford & Palme, 93) and takes a top-down approach to the development of group communication system services by ascertaining needs in group communication applications. The research has been carried out in three phases.

The first phase involved carrying out case studies considering three group editing activities: a Newsletter, a Technical Paper and a Software Development Team. The Newsletter activity has been discussed in depth and used for the prototype. The other two activities have been used as supporting (minor) cases studies. The case studies have determined the underlying needs for these activities.

The second phase involved the development and testing of a working prototype for editing a Newsletter.

The third phase involved modelling of group editing activities based on common issues determined from the previous phase, and has led to recommendations for the services and tools required for group editing activities. Recommendations have been proposed to modify/extend the functionality of the X.420 recommendation (Interpersonal Messaging System).

The research has established that the main issues are to provide version and access control, to work out the logical structure of the common store, to construct the group editing header components and to provide management of the activity.

This concluding chapter also evaluates the 'SAGE' project. The evaluation discusses observations made during the 'SAGE' project research, and considers advantages and disadvantages of the major recommendations. Finally novel features of the research are listed and suggestions for future research in the field given.

11.1 Observations on Research

11.1.1 General observations

To start the work it is necessary to define the problem. The 'SAGE' project is defined in relation to the OSI network reference model and the X.400 recommendations, which proved a good foundation and placed no restrictions on the development. The X.400 implementation used provided many of the facilities needed for the project, and its protocol extension facilities allowed successful provision of the required functionalities for Group Editing.

There is a considerable amount of literature relevant to the 'SAGE' project. The most useful proved to be either standards, or work either leading to, or oriented towards, standards. The 'SAGE' project builds directly on work described in the literature and/or standards. The 'SAGE' project does not seek to comment on standards but is concerned with the use of existing standards and making recommendations for extensions to X.400 for group editing activities.

The research method adopted in the 'SAGE' project led to satisfactory results. The research method used case studies, prototyping and modelling of group activities. The case studies are based on interviews with users. Three related cases on a particular topic (i.e. group editing) were chosen in order to deduce common functionality features to derive a model for a set of group editing activities.

The prototype was based on the specifications extracted from the case studies, and was specific to the Newsletter activity. The prototyping approach provided clarification of the implementation and shape of the services, and was used as a testbed and as a vehicle for user comments to improve the final functionality features.

The modelling of group activities allowed the determination of the common services and tools for a set of activities. This led to the classification of group activities into three application groups which assist construction of framework models for a wide range of group activities.

11.1.2 Specific Observations

11.1.2.1. Case studies

The choice of users for interview in the case studies was spread over different people in different activities, giving a breadth of user views over the three activities used for case study. All users had working experience of their activities. All the users interviewed

were from a University environment, but this made them accessible for detailed discussion and led to their having a good understanding of the work and to willing cooperation.

There is always a question where using case studies of how many people to interview, how suitable they are and whether the right topics were chosen. In the context of this work the case studies provided sufficient material to advance the work.

The choice of verbal interaction to establish the users views on editing issues proved successful: the verbal format meant that the user was less constrained by the question, and gave full answers that often extended beyond the question. Cross questioning was made easy by the verbal format.

The issues presented to the user at interview were extracted from the literature. The issues were perceived by the users as sensible and relevant, and were sufficiently general to allow the user to express their views without being too broad to give a clear focus. The set of issues raised with the user provided sufficient information to form the initial specification.

The case studies chosen covered different functionalities within Group Editing. Three cases were chosen as sufficiently different to cover most functionalities needed in the Group Editing environment. The specific choices were guided by the availability of users for interview. Review of other Group Editing activities confirmed that the three cases chosen had been sufficiently broad.

The method of case studies and the case study chosen proved sufficient for establishing a adequate set of requirements for group editing and to provide a framework for the prototype and to permit the construction of good specifications.

11.1.2.2. Prototyping

A throw-away prototype was developed to demonstrate the proposed functionality to the users. The demonstration of the prototype proved useful in obtaining new ideas from users on the functionality they required.

The prototype was designed to fit in with structures currently proposed for Group Communications. The trend of research suggests that Group Editing requires identification of advanced services and knowledge-based Information Handling beyond the basic services already defined in X.gc. The implementation of the prototype in three modules provides a structure consistent with this approach: parts of the Activity Generator and Activity Responder modules provide the advanced services, while the Activity Monitor provides the Information Handling functionality.

The prototype was developed with modular program design. It may have been better to use an object oriented approach for the prototype which would have been closer to the proposed group communication information model. It could also have been useful to include X.500 services within the prototype. However t he complexity of these approaches would be more suited to a final implementation than to a prototype designed to evaluate functionality.

The demonstration of the prototype was to fewer users than had contributed to the first set of interviews. This was caused by movement of some of the users. It would have been better to have all the initial set of users for the demonstration, but those who were available for the demonstration were still sufficiently diverse to provide information over the range of activities.

Most of the issues raised in the initial interview were covered again in the demonstration. The users suggested some further functionality needs after seeing the

demonstration. The users were satisfied that the demonstration covered the main functionalities, and the information obtained from the demonstration provided confirmation that the specification had been correctly established.

The lack of error handling on the prototype required that it be demonstrated to the user rather than allowing the user to 'play' with the prototype. This may have restricted the range of user responses, but despite this the range of issues raised during the demonstration was extensive and covered a wide range of functionality features. The long duration of the demonstration made it better to allow the user to write notes as the demonstration proceeded, and it would have been difficult to keep up the continuity of the demonstration if verbal answers had to be recorded by the demonstrator.

11.1.2.3. Modelling of Activities

The modelling of group activities in the 'SAGE' project provides a structural approach to extend the group communication framework model. The modelling also proposes the three number of models to covers a wide range of group activities. But there are some advantages and disadvantages, which are discussed in following paragraphs.

The 'SAGE' project proposes modelling of a wide range of group activities as three classes of application groups: informative, supportive and objective application groups. Some advantages and disadvantages of this classification are listed, then discussed in the following paragraphs.

Advantages:

- Modelling of all group communication activities into three application groups,
- Three sets of services and tools will provide functionality for all group activities,
- Reduced number of service agents needed in a group communication system,
- Provides hierarchic structure for group activities in OSI environment,
- Classes do not restrict use of an operational object to any one class.

Disadvantages:

- Fitting of activities into groups:

  May be difficult to place all group activities into these three groups,

  An activity may fit into more than one of the three group,

- Wider class functionality increases complexity for service agents,
- Complexity may arise in forming relationships between operational objects and information objects.

The proposed classification implies that there will be only three general models to cover all group activities. This will allow concentration of effort on refining these models. Modelling of future activities will be simplified if they can be included within one of the groups.

Rather than developing individual services and tools for each group activity, three universal tool sets would be all that is required to provide functionality for all group communication activities. For example, a 'Help Desk Facility' activity (Benford, 91) would fall within the Informative group, and is similar to 'distance learning' or 'bulletin board' activities. The functional requirement of these activities are similar and require similar services and tools.

The classification provides a structure that allows an individual activity to be recognised as part of an OSI system. As further group communication applications are developed in the future, there is possibility of a proliferation of service agents being developed to meet the needs of each application. By using the proposed classification of application groups developers will be encouraged to work with the service agents for that group, limiting the number of service agents to three. There is already an increasing number of group communication service agents (e.g. computer conferencing service agent, EDI service agent etc.) in the group communication system.

The proposed classes do not restrict the use of operational objects to any one class. Much of the functionality of each class can be made common. The common set of operational objects can serve all three classes. There are some special tools for each class and only the special tools for the class in use needs to be available. For example analysis of the Objective application group shows that these activities have common functionality with the supportive application group. Most of the editing services are common for the objective application group. For example, an executive board needs to prepare minutes of meetings, which could be prepared with the editing services and tools. However the objective application group has special needs, for example 'voting' or 'opinion analysis' or similar decision making functionality.

Clearly there are several advantages in the proposed classification of group communication activities. There are however, some disadvantages.

It may not be possible to place group activities that arise in the future into one of these classes, or perhaps an activity will fit into more than one application group. For example the date planning activity (Prinz & Woitass, 89) will use the supportive application group services in most situations, and therefore would appear to belong in the supportive application group. However, if a date could not be agreed, then the services of the objective application group are needed.

By implementing functionality for an entire class of activities in one service agent the complexity of the service agent is increased. The complexity of relationship between operational objects and information objects will also be increased. For example a relationship such as 'add member' must be linked to all activities in the activity class to provide service to them.

The proposed classification brings significant structural advantages. The disadvantages lie at the implementation level. The implementation difficulties are not great and are a price worth paying for the major structural advantages.

11.1.2.4. Recommended Services and Tools

The proposed services and tools (described in section 10.1.2 in chapter 10) have some advantages and disadvantages.

The services and tools are easy to implement as they are supported by existing standards and implemented in a Unix and X.400 environment. The services are described in the same style as CCITT draft X.gc, making it easier to add these tools to existing standards. The services and tools proposed by the 'SAGE' project are compatible with the classification of activities proposed in the research.

However the proposed tools require a different header component files for most of operations, which must be created and stored. The proposed tools will be sufficient to meet the requirement of most group editing activities, but additional functionality may be needed for some activities in the application group. A group communication service agent (GCSA) which uses the basic services of X.gc and provides advanced services and an information handling system in one service agent may be difficult to specify and implement. The complexity may make it difficult to incorporate the functionality as a modification of X.420.

The main strength of the proposal is the integration with existing standards, which will ensure strong interworking.

The disadvantages lie more in the implementation and can be overcome. However the question of how well the proposed modification will fit into X.420 requires further considerations.

11.1.2.5. Additional Functionality

The following additional functions are proposed (section 6.2.2.7 in chapter 6) which were implemented in the prototype:

Reminder facility: when a contribution is not submitted within a specific time the system generates an automatic reminder to the author.

Delivery failure check facility: if a contribution is submitted to the group editing organiser, but returned to the author for any reason, then the contribution is automatically located and re-sent.

The automatic reminder facility worked well, but users felt that there should be a manual reminder facility in addition.

The delivery failure check proved satisfactory, but could be improved by having a double check. Firstly, it should check delivery failure and if there is a delivery failure, then the contribution should be re-sent. To improve the procedure, arrival of the contribution at its destination (i.e. at common store) should generate and send a 'contribution arrival' notification to the originator. If an arrival notification does not arrive within a specific time, a 'contribution check' event should be generated automatically to check on behalf of the originator, the arrival of the contribution at the common store. While it would not be a foolproof system it would increase the credibility of the delivery check.

11.2 The 'SAGE' Project Recommendations

The 'SAGE' project proposes the **classification** of group communication activities into **three innovative application groups** (i.e. **Informative** group, **Objective** group and **Supportive** group), which will need new application types each with their own service agents.

The 'SAGE' project view on the functionality of any group activity is that it can be divided into two parts: the **tools part** and the **services part**. The first part comprises the operations directly concerned with the users of the group. The second part is the underlying functionality required to meet such operations. The underlying functionality of these tools is to be provided by an application service agent.

Abstract Service Definitions are given for fourteen proposed new operations (e.g. Create-Activity, Update-Document, Accept-Contribution and Edit-Notification) which are in addition to the CCITT working document X.gc (Joint ISO/IEC/CCITT, 91). The Abstract Service Definitions are given in the style of X.gc.

It is also proposed that the group editing services are modelled onto the message handling system and require different header parameters which is achieved with the help of the protocol extension facilities provided in X.420 recommendation. This extends the protocol functionality of the X.420 recommendation.

11.3 The Novel Features of This Research

1. Classification of group communication activities within three application groups.
2. Proposal of three application groups: **Informative**, **Supportive** and **Objective** application groups.
3. Proposal to define a new Group Identifier for each application group so it can be recognised as an OSI application.

4. Proposal to incorporate all basic services, advanced services and rule-based information handling for each application group agent into a service agent.
5. Derivation of Abstract Service Definition for group editing services operations.
6. Proposal of extension and modification of X.420 fields for the group editing services.
7. Mapping of group editing services onto the message handling system and proposal of new component files for such services.
8. Proposal of group editing services and tools, in addition to CCITT.

11.4 Future Work

Work in the area of group communication spreading rapidly. At the same time the use of standard protocols is also increasing widely (i.e. X.400 and X.500). This is the right time to set up a logical approach to work out the strategy for modelling group communication activities onto the existing standard protocols. Based on the work described here, the following research is now needed.

Categorisation of group activities into three application groups - In view of the large number of group activities, it would be useful to categorise the known group activities into three application groups. It would be helpful to have as many group activities as possible classified within the groups at an initial stage. This would allow application groups to be designed with maximum functionality.

Investigation of the common functionality within each group - The classes of the application groups are based on similar functionality features. Further work needs to be done on identifying common functionality within each application group. It is proposed to start with the Informative application group which has the least functionality requirements in terms of operations and complexity of operations.

Common functionality between application groups - The common services and tools of each application group should be mapped onto all three (Informative, Supportive and Objective) application groups. This would enable a common set of services and tools for a wide range of group activities to be derived. The services specific to each application group would be built within that application group.

Definition of the services and tools for each application group - The services and tools required by each application group should be defined as the services provided by the respective application service agent. For example the services and tools required by the Informative application group should be recognised by the Informative Application Group Agent, and likewise by the Supportive Application Group Agent for that group. Each service agent would carry the list of activities to be served by that application group agent.

Extension of the CCITT framework model for the application groups - The 'SAGE' project proposes the extension of the framework model up to the level of each application group but has not implemented it. Therefore there would be three framework model to cover all group communication activities. Each framework would have common built-in functionality containing basic and advanced services and information handling requirements, as per the requirement of each application group.

Modelling of individual activities onto the framework model - The individual activities are to be modelled onto their respective application group's framework model. The features specific to the activity concerned would be built on top of the framework. However the common services and tools required by the activity would be provided by the service agent.

11.5 Conclusion

The 'SAGE' project has investigated the needs of a group of people working cooperatively in an OSI environment, and recommended tools and services to meet these needs.

The project used case studies to identify user requirements and to determine common functionalities for a variety of group editing activities. A prototype was implemented in an X.400 environment and proved useful in refining user requirements and testing the proposed functionalities.

The conceptual modelling follows current CCITT proposals, but a new classification of group activities has been proposed: Informative, Objective and Supportive application groups. Each of these application groups should have their own Service Agent. Use of this classification allows the possibility of developing three sets of tools which will cover a wide range of group activities, rather than developing tools for individual activities. Group editing has been considered to be in the Supportive application group.

A set of additional services and tools to support group editing are proposed in the context of the CCITT draft on group communication, X.gc. The proposed services and tools have been mapped onto the X.400 series of recommendations, with the Abstract Service Definition of the operational objects defined, along with their associated component files, by extending the X.420 protocol functionality.

It has been proposed that each of the Informative, Objective and Supportive application groups should be implemented as a modified X.420 inter-personal messaging system.

References

Avison, D.E., (1985), The Open Book, Information Systems Development: A Data Base Approach, UK: Blackwell Scientific Publications.

Bartlett, K.A., (1986), " Standards for communications-too many or too few ", in Strokes, A.V. Dr. (ed) *Communication Standards: State of the Art Report* , England: Pergamon Infotech, pp.3-9.

Benford, S., (1989), "The Global Information Space", in Smith, H., Onions, J. & Benford, S., (eds), *Distributed Group Communication the AMIGO information model*, Ellis Horwood Ltd Publishers, Chichester.

Benford, S. (1991), "Building Group Communication on OSI",*Computers Networks and ISDN Systems*, Vol./Part 23(1-3) pp. 87-90.

Benford, S., Howidy, H., Shepherd, A. & Smith, H., (1990), "Group Communication in an Open Systems Environment ",*The Grace Project Phase I Report*, Research Communication Group: Nottingham University, UK.

Benford, S., Palme, J., (1993), "A Standard for OSI Group Communication",*Computers Networks and ISDN Systems,Vol./Part 14(5-6)*, pp. 933-946.

Benford, S., Turoff, M. & Palme, J., (1992), "An OSI Standard to Support Asynchronous Group Communication",*Computers Standards & Interfaces, Vol./Part 25(8)*, Elsevier Science Publishers B.V.: North-Holland, pp. 363-373.

Black, U., (1991), The X Series Recommendations Protocols for Data Communications Networks, USA: McGraw-Hill Inc.

CCITT/ISO Recommendations, (1988), "X.500: Data Communication Networks Directory X.500-X.521 Volume VIII", (Blue Book).

CCITT/ISO Recommendations, (1988), "X.400: Data Communication Networks Message Handling Systems X.400-X.420 Volume VIII", (Blue Book).

Checkland, P., (1990), "Soft System Methodology in Action", John Wiley & Sons Ltd., Chichester, England.

Cross, John A., (February, 1989), " Technology Transfer in the 1990's: Evaluating Computer-Support for Cooperative Work ",*Computing Trends in 1990's 17th Annual ACM Computer Science Conference.*, New York: ACM Press, pp. 474.

Jakobs, K., (1991), "Beyond the interface - Group Communication Services Supporting CSCW", in Venijan-Stuart, A.A., Sol, H.G. & Hammersley, P., (eds),*Support Functionality in Office Environment*, Elsevier Science Publishers B.V.: North-Holland, pp. 67-84.

Jarratt, P., (1989), Birmingham University, "Distributed Computing and Networks ", M.Sc. lecture notes Sept/Oct.

Joint ISO/IEC/CCITT, (1991), of the Group Communication Working Document X.gc Version 7, *CCITT Study Group I/Q.15, VII/Q.18 and ISO/IEC JTC 1/SC 18/WG 4 SPW Messaging*, Period 1989-1992 Document **MH-534**.

Kille, S.E., (1991), Implemenating X.400 and X.500: The PP and QUIPU Systems, USA: Artech House, Inc.

Klehn, N. & Kuna, M., (1991), "CSCW and Group Communication", *Informatik, Informationen Reporte*, Vol./Part 4, pp. 7-16.

Kuna, M. & Klehn, N., (1991), "A Three Level Approach to Model Systems for Computer Supported Cooperative Work", *Informatik, Informationen Reporte*, Vol./Part 4, pp. 222-231.

Lanceros, A. G. & Saras, J. A., (1988), " Group Communication Support in the MHS Environment ", in Speth, R. (ed), *EUTECO '88' on Research into Networks and Distributed Applications*, North-Holland, pp. 311-322.

Lanceros, A & Saras, J., (1989), "The Distribution Service", in Smith, H., Onions, J. & Benford, S. (eds), *Distributed Group Communication the AMIGO information model*, Ellis Horwood Ltd Publishers, Chichester.

Manros, C-U., (1989), The X.400 Blue Book Companion, UK: Dotesios Pronters Ltd.

Marshall, T.Rose, (1989), The Open Book, A practical Perspective On OSI, USA: Prentice Hall Inc.

Medina, M., Maude, T. & Smith, H., (October, 1986), " X.400 and Group Communication Support ", *International Business Strategy Conference,* Online Publication, pp. 261-269.

Nunez, M., (1989), "Archive Services", in Smith, H., Onions, J. & Benford, S. (eds), *Distributed Group Communication the AMIGO information model*, Ellis Horwood Ltd Publishers, Chichester.

Olson, G. M., (1989), "The Nature of Group Work ", *Proceeding of the Human Factors Society 33rd Annual meeting*, pp. 852-855.

Palme, J., (1986), "Group Communication & its role in Electronic Mail ", *Electronic Message Systems*, Online Publications: Pinner, UK, pp. 297-307.

Palme, J., (1987), "Distributed Computer Conferencing ", *Computer Networks and ISDN Systems* , Vol./part 14(2-5), North Holland, pp. 137-145.

Palme, J., (1988), " Extending Message Handling to Computer Conferencing ", in Raviv, J. (ed),*Computer Communication Technologies for the 90's*, Elsevier Science Publishers B.V.: North-Holland, pp. 44-59.

Peek, J.D., (1991), MH & xmh E-mail for Users & Programmers, USA: O'Reilly & Associates, INC.

Plattner, B., Lanz, C., Lubich, H., Muller, M. & Walter, T., (1991), X.400 Message Handling Standards, Interworking, Applications, Addison-Wesley Publishing Company Inc.

Prinz, W. & Woitass, M., (1989), " Date Planning System - Example of a Cooperative Group Activity ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 359-379.

Roechr, W., (1986), " Message Handling Protocols " in Strokes, A.V. Dr. (ed) *Communication Standards: State of the Art Report* , England: Pergamon Infotech, pp.187-202.

Schicker, P., (1989), " Message Handling Systems, X.400 (An Overview)", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 3-41.

Sherif, M.H. & Sparrell, D., (1992), "Standards and Innovation in Telecommunications", *IEEE Communications Magazine*, July 1992, pp. 22-28.

Smith, H., (1989), "An Introduction to Group Communication Service Requirements", in Smith, H., Onions, J. & Benford, S. (eds), *Distributed Group Communication the AMIGO information model*, Ellis Horwood Ltd Publishers, Chichester.

Smith, H., Onions, J. & Benford, S., (1989), *Distributed Group Communication the AMIGO information model*, Ellis Horwood Ltd Publishers, Chichester.

Sommerville, I., (1989), Software Engineering (Third Edition), Addison-Weley Publishers Ltd.

Steedman, D., (1986) " Message Handling Systems " in Strokes, A.V. Dr. (ed) *Communication Standards: State of the Art Report* , England: Pergamon Infotech, pp.217-231.

Strokes, A.V. (ed) , (1986), " The OSI model of Open System Interconnecting ", *Communication Standards: State of the Art Report* , England: Pergamon Infotech, pp. 315-318.

Tanenbaum, A. S., (1989), *Computer Networks* (Second Edition), USA: Prentice-Hall Inc.

Taylor, J.M. (January, 1990), " Co-operative Computing and Control ", *IEE Proceedings*, Vol.137, Pt. E, No.1, UK, pp. 1-16.

Wagner, B. & Palme, J., (1989), "Support for Advanced Group Communication", in Smith, H., Onions, J. & Benford, S. (eds), *Distributed Group Communication the AMIGO information model*, Ellis Horwood Ltd Publishers, Chichester.

Weiss, K-H. & Bogen, M., (1989), "A Group Communication Service Architecture", in Smith, H., Onions, J. & Benford, S. (eds), *Distributed Group Communication the AMIGO information model*, Ellis Horwood Ltd Publishers, Chichester.

Weiss, K-H., (1989), "Use of the Directory Service for Group Communication Support", in Smith, H., Onions, J. & Benford, S. (eds), *Distributed Group Communication the AMIGO information model*, Ellis Horwood Ltd Publishers, Chichester.

Wilson, P., (1991), Computer Supported Cooperative Work, Great Britain: Intellect Books Publishers.

Bibliography

Ahuja, S.R. & Ensor, J.R., (1992), "Coordination and Control of Multimedia Conferencing", *IEEE Communications magazine*, Vol. 30 No. 5 pp. 38-43.

Badine S.A., Kintzig C. & Okoko, M., (1988), "A Job Submission Service within the Message Handling System", in Boyanov & Angelinov (eds), *Network Information Processing Systems*, Elsevier Science Publishers B.V.: North-Holland, pp. 265-279.

Benford, S., (1989), " Navigation and Knowledge Management within a Distributed Directory System ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 143-161.

Christensen, W. & Suess, R., (1978), "Hobbyist Computerized Bulletin Board", *Byte Oublication Inc, November 1978*, pp. 150-157.

Clark, B. & O'Donnell, S., (1991), "Computer Supported Cooperative Work", *British Telecom Technol Journal, January 1991*, Vol./Part 9(1) pp. 47-55.

Clark, W.J., (1992), "Multipoint Multimedia Conferencing", *IEEE Communications magazine*, Vol. 30 No. 5 pp. 44-50.

Cole, R. & Burns J., (1989), "An Architecture for Mobile OSI Mail Access System", *IEEE Journal on Selected Areas in Cmmunications, Vol.7 No. 2, February 1989*, pp. 249-256.

Cole, R.H., Hall, C.S., Rush, P.D.C. & Walker, J., (1988), "Mobile Access to Secure Inter-Personsal Messaging", *International Open Systems Coference 88, Pinner, On-Line Publications*, pp. 19-21.

Cornell, P., (1989), "Ergonomic Environment Aspects of Computer Supported Cooperative Work", *Proceedings of the HUMAN FACTORS SOCIETY 33rd Annual Meeting*, pp. 862-866.

Danthine, A. & Godelaine, P., (1989), " MHS in a Corporate Communication System Offering InterNet Service ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 305-319.

Dede, C.J., (1990), " The Evaluation of Distance Learning: Technology-Mediated Interactive Learning ", *Journal of Research on Computing in Education*, Vol./Part 22(3), Florida, pp. 247-264.

Dittrich, J. & Wolisz, A., (1992), "Toward Cooperative Use of Shared Data in Open Distributed Systems", in Meer, J.de, Heymer, V. & Roth, R, *Open Distributed Processing*, Elsevier Science Publishers B.V.: North-Holland, pp. 179-190.

Eliasen, F., Nordli, I. & Danielsen, T., (1989), "Communication via Shared Conceptual Memory", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Message Handling Systems and Distributed Applications* , Elsevier Science Publishers B.V.: North-Holland, pp. 383-413.

Ellis, C.A., Gibbs, S.J. & Rein, G., (1990), "Design and Use of a Group Editor", in Kockton, G. (ed), Engineering for Human-Computer Interaction, Elsevier Science Publishers B.V.: North-Holland, pp. 13-25.

Engelbart, D. & Lehtman, H., (1988), "In Depth Groupware: Working Together ", *Byte December 1988*, pp. 245-252.

Gardner, A., (1986), " British Telecom's X.400 service ", *Electronic Message Systems*, Online Publications: Pinner, UK, pp. 345-356.

Genilloud, G. (April, 1990), " X.400 MHS: First step Towards an EDI Communications Standard ", *Computer Communication Review 1990*, Vol. 20(2), pp. 72-86.

Golfer, J.L., (1985), "Going Online with America: The World of Electronic Board", Online 85 Conference, Weston, Online Inc: pages 133-137.

Halsall, F., (1989), Data Communications, Computer Networks and OSI (Second Edition), Addison-Wesley Publishing Company Inc.

Handley, M.J. & Wilbur, S., (1992), "Multimedia Conferencing from Prototype to National Pilot", e-mail: M.Handley@cs.ucl.ac.uk & S.Wilbur@cs.ucl.ac.uk.

Hansen, A., (1988), "RARE MHS project Review", *Computer Networks and ISDN Systems*, Elsevier Science Publishers B.V.: North-Holland, pp. 8-12.

Henshall, J. & Shaw, S., (1988), " Message handling systems, X.400/MOTIS ", *OSI Explained: end-to-end computer Communication Standards* , Chichester: Ellis Horwood, pp.158-200.

Herbert, D., (1990), "EDI and Communications - the technology debate", *EDI Analysis January 1990*, pp. 10-11.

Hiltz S. R., & Turoff Murray, (1985), "Structuring Computer-Mediated CommunicationSystem to avoid Information Overload", *Comunications of the ACM*, Volume 28 number 27 pp. 680-689.

Hiltz S., R., Turoff M. & Johnson K., (1989), "*Decision Support Systems*", Elsevier Science Publisher B.V.: North Holland, pp. 217- 232.

Hunter, R. & Cross, B.A., (1991), "ODA- an Open Information Architecture for Interconnecting Office Systems, *British Telecom Technol Journal, January 1991*,

Kenny, P., (1988), " Unified Implementation of X.400 and EDI ", *Electronic I information Interchange -X400 and EDI, IEE Colloquium on X400 and EDI*, London: IEE, pp. 9/1-9/3.

Kille, S.E., (1989), " PP- A Message Transfer Agent ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 115-127.

Kille, S.E., (1989), " The Quipu Directory Service ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 115-127.

Kime, H. & Smith, Jr., (1992), "Accessing Multimedia Network Services", *IEEE Communications magazine*, Vol. 30 No. 5 pp. 72-80.

Krcmar, H.A.O., (1991), "Computer Supported Cooperative Work - State of the Art", *in Bullinger H.J. (ed), Human Aspects in Computing: Design and Use of Interactive Systems and Information Management*, Elsevier Science Publishers B.V., pp. 1113-1117.

Kretz, F. & Colaitis, F., (1992), "Standardizing Hypermedia Information Objects", *IEEE Communications magazine*, Vol. 30 No. 5 pp. 60-70.

Lanceros, A. G. & S., Juan A., (1989), " Definition of Group Communication Facilities in the MHS", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 323-336.

Lawrence, A., (1989), Grand Plans (X.400 Message Handling Standards), *Communicate June 1989,* , pp. 46-48.

Lebeck, S. & Lubich, H., (1989), " Final Report: Multi-Media Messaging ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 547-550.

Lebeck, S.K., (1989), " Implementing MHS: 1984 versus 1988 ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 115-127.

Lebeck S. K., (1989), " Implementing MHS: 1984 versus 1988 ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 101-113.

Leland, M.D.P., Fish, R.S. & Kraut, R.E., (1988), "Collaborative Document Production Using Quilt", *ACM SIGOIS Conference 1988*, pp. 30-37.

Lubich, H. & Plattner, B., (1988), "Naming and Addressing in SWITHmail", *Computer Networks and ISDN Systems*, Elsevier Science Publishers B.V.: North-Holland, pp. 48-54.

Lubich, H. & Plattner B., (1990), "A Proposed Model and Functionality Definition for a Collaborative Editing and Conferencing System", in Gibbs, S. & Venijan-Stuart, A.A. (eds), *Multi-User Interfaces and Applications*, Elsevier Science Publishers B.V.: North-Holland, pp. 215-232.

Lubich, H.P., (1990), "MultimETH, a Collaborative Editing and Conferencing Project", *Computers Networks and ISDN Systems*, Vol./Part 19(3-5), Elsevier Science Publishers B.V.: North-Holland, pp. 215-223.

Mahon, B., (1989), "Electronic Mail System, ", *Electronic Transfer of Information and its Impact on Aerospace Development*, Neuilly, France 1990 AGARD, pp. 12

Mantei Marilyn, (1989), "Observation of Executives Using a Computer Supported Meeting Environment", *Decision Supprt Systems*, Elsevier Science Publishers B.V.: North-Holland, pp. 153-166.

Mark, L. A., (1989), " Technology for Computer-Supported Meeting ", *Proceeding of the Human Factors Society 33rd Annual meeting*, pp. 857-861.

Marsden, B.W., (1991), Communication Network Protocols (3rd Edition) OSI Explained, Sweden: Chartwell-Bratt (Publishing and Training) Ltd.

Mendoza, E., (1988), "Directory Services and COSINE", *omputer Networks and ISDN Systems*, Elsevier Science Publishers B.V.: North-Holland, pp. 44-47.

Mount, R.P., (1988), "What Users Want", *Computer Networks and ISDN Systems*, Elsevier Science Publishers B.V.: North-Holland, pp. 146-149.

Ngoh, L.H. & Hopkins, T.P., (1990), " Initial Experience Implementing Multicast Facilities in Computing Supported Cooperative Work ", UK IT 1990 Conference, London: IEE, pp. 196-203.

Olsen, B.A., (1986), "Cost benefit analtsis of mail & conference systems", *Electronic Message Systems*, Online Publications: Pinner, UK, pp. 147-164.

Open Systems Data Transfer, (October 1989), "EDI X.400: Projects and Prospects",*Omnicom Newsletter service* , Omnicom International Ltd. UK.

Palme, J., (1986), "Data Bases in Computer Messaging Systems", *8th International Conference on Computer Communications*, pp. 67-71.

Palme, J. (1986), " Data bases in Computer Messaging Systems ", *New Communication Services: A Challenge to Comp. Technology*, Amsterdam: Elsevier Publishers, North-Holland, pp. 67-71.

Pays, P.A. & You, Y.Z., (1989), " A General Multi-User Message Store ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 399-414.

Racke W. F. & Fischer K., (1988), "Extending an Existing Mail Service to Support X.400 Message Handling", *10th Computer Networking Symposium*, IEEE Computer Sco. Press: pages 245-53.

Rafaeli S., (1986), " Electronic Bulletin Board, A Computer Driven Mass Medium", *Computers and Social Sciences*, Vol 2(3), Paradingm Press Inc., pp. 123-136.

Santo, H., (1988), " Advanced Messaging in Groups ", *Interim Report*, North-Holland, pp. 55-60.

Scheller A. & Fuhrhop C., (1990), "Joint Editing with DAPHNE", in Vandoni, C.E. & Duce, D.A. (eds), EUROGRAPHICS '90, Elsevier Science Publishers B.V.: North-Holland, pp. 113-124.

Schulze G., (1988), " Office Document Architecture and Its use in Message Handling Systems", in Raviv, J. (ed),*Computer Communication Technologies for the 90's*, Elsevier Science Publishers B.V.: North-Holland, pp. 38-43.

Shillingford, J., (1990), "Electronic Messaging in the 1990s", *Communicate*, Vol./Part pp. 42-44.

Shoshana, L., (1992), "Delivering Interactive Multimedia Documents over Networks", *IEEE Communications magazine*, Vol. 30 No. 5 pp. 52-59.

Touillet, D., (1988), "X.400 the Key for new Networking Applications the example of Atlas 400", in Raviv, J. (ed),*Computer Communication Technologies for the 90's*, Elsevier Science Publishers B.V.: North-Holland, pp. 470-475.

Valentine, I., (1988), "What Next After X.400", *Communication International January 1988*, pp. 29-30.

Wagner, B., Bogen, M., Nunez, M., Gallardo, J., Benford, S., Onions, J. & Palme, J., (1989), "Piloting", in Smith, H., Onions, J. & Benford, S. (eds), *Distributed Group Communication the AMIGO information model*, Ellis Horwood Ltd Publishers, Chichester.

Wilbur, S., (1989), " Group Communication as a Tool in an Organisation Support Environment ", in Stefferud, E., Jacobsen, O.J. & Schicker (eds), *Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems and Distributed Applications* Costa Mesa, California: North Holland, pp. 337-344.

Wilson, P., (1988), "Key Research in Computer Supported Cooperative Work (CSCW)", *Research into Networks and Distributed Applications*, Elsevier Science Publishers B.V.: North-Holland, pp. 211-226.

Yamazaki, K. & Suzuki, H., (1984), "A Methodology of Project Management for Micro-Computer System Development", *Journal of Japan Society of Precision Engineering Vol./Part 50(3)*, pp. 550-555.

Appendix 1

Overview of Seven Layers

A 1.1 Overview of Seven Layers

The brief functional description of seven layer in the OSI reference model given below:

(i) The **Physical Layer** (Layer 1)

The Physical Layer is the lowest layer of the ISO model and interfaces to the actual communication medium. This layer provides mechanical, electrical, functional and procedural characteristics to establish, maintain and release physical connections between two data link entities. It handles the encoding of data into signals compatible with the medium, bit timing and modulation standards.

(ii) The **Data Link Layer** (Layer 2)

The Data Link Layer builds upon the services offered by the Physical Layer in order to transmit a stream of bits without error. Layer 2 synchronises transmission and handles error control and recovery so that information can be transmitted over the Physical Layer. It establishes an error-free communication path between network nodes over the Physical Layer channel, message access to the communication channel, and ensures the proper sequence and integrity of transmitted pockets of data. For example, an OSI standard known as High-Level Data Link Control (HDLC) provides the data-link connect procedures (Jarratt, 1989).

The **Network Layer** (Layer 3)

The Network Layer provides a service which is independent of the actual network(s) being used. The Network Layer establishes, maintains and terminates communication between nodes. It sets-up the most economical path, both physical and logical, between communicating nodes, routes messages through intervening nodes to their destination and controls the flow of messages between nodes. Any relay functions are hop-by-hop service enhancement protocols used to support the network service between the OSI end open systems are operating below the Transport Layer, i.e. within the Network Layer or below.

The **Transport Layer** (Layer 4)

The Transport Layer provides a service to transport data from one system to another without concern for the underlying network(s). It provides end-to-end (source-node-to-destination-node) control of a communication session once the path has been established, allowing processes to exchange data reliably and sequentially, independent of which systems are communicating, and their location in the network. It is basically acts as the liaison between user and network. It is this layer which provides end-to-end sequence control, error detection, error recovery, flow control and monitoring & supervisory functions.

The **Session Layer** (Layer 5)

The Session Layer is concerned with the maintenance of logical sessions. The Session Layer manages dialogue, establishing and controlling system-dependents of communications sessions between specific nodes on the network. A node is any intelligent unit on a network that has an integral central processing unit (CPU). Every node is uniquely addressed on the network (Jarratt, 1989).

The **Presentation Layer** (Layer 6)

As the name implies, the Presentation Layer is responsible for presenting data to entities in the Application Layer in the form that they understand. The Presentation Layer resolves the difference of varying data formats between systems of different vendors. It works by transferring data in a system independent manner, performing appropriate conversions at each system.

The **Application Layer** (Layer 7)

The Application Layer is the highest layer in the ISO model and it can be considered as the layer in which all application processes reside. The Application Layer provides information services to support end-user application tasks such as file transfer, remote file access database management, electronic mail and terminal access. It manages communications between applications.

Appendix 2

Suplementary Information on MHS

A 2.1 Protocols "P7", "P3" and "P1"

The structure of access protocol "P7" views between UA and the MS is shown in Figure A 2.1. Figure A 2.2 shows the communication between MTS and MS or UA using "P3" protocol. The service elements specific to protocols "P3" and "P7" cater for the submission of messages or probes and the delivery of messages or reports as well as the administration of the connections (CCITT, 88). The protocol "P7" is identical to the protocol "P3" except the delivery port services are replaced by the retrieval port services. The protocols "P3" and "P7" in addition make use of the remote operation service element (ROSE). ROSE is a simple protocol that triggers an action in the partner MTA. Such an action is for example the receipt of a message. The action always responds to the initiator with either a result or an error message.

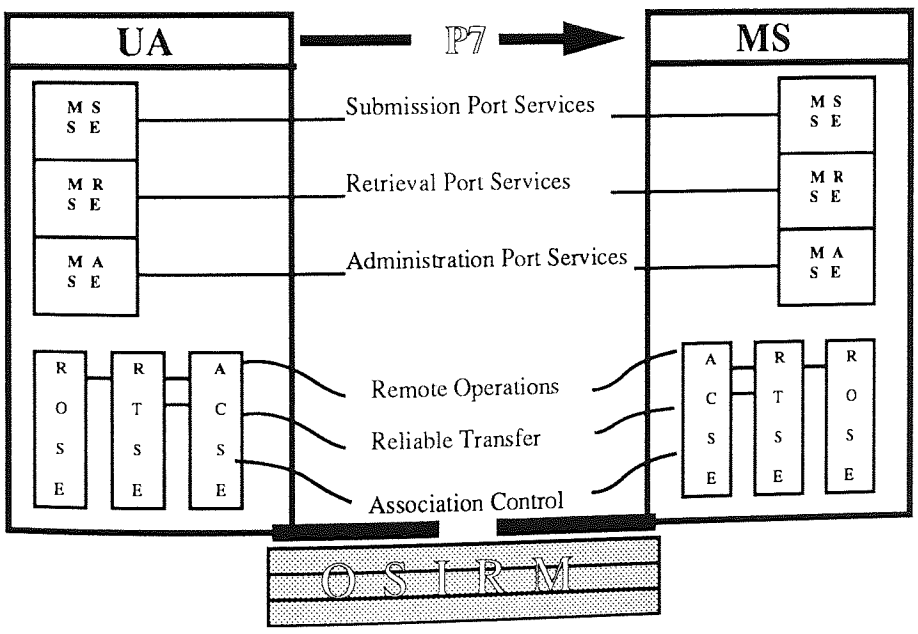


Figure A 2.1 Structure of the P7 protocol.

The structure of two MTAs communicating with protocol "P1" is shown in figures A 2.3. The message transfer service element (MTSE) utilizes the reliable transfer service element (RTSE) which in turn makes use of the association control service element (ACSE). ACSE

is responsible for the establishment and release of a connection. Both RTSE and ACSE depend on the services of the presentation layer, i.e.. layer 6 of the OSI model. The reliable transfer service element hides the particulars of the OSI protocols. The MTSE hands over to the RTSE a complete message. This message is then segmented by RTSE and transmitted to the partner MTA through OSI layer services. The RTSE ensures that those services are employed correctly, so that even in the case of a complete disruption of the connection, the message, probe or report is not lost nor replicated.

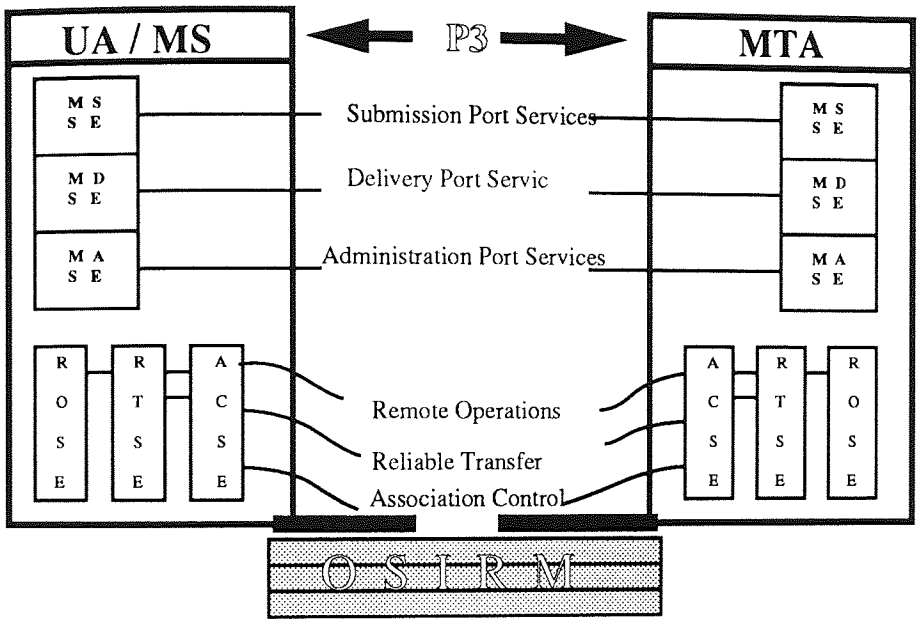


Figure A 2.2 Structure of the P3 protocol.

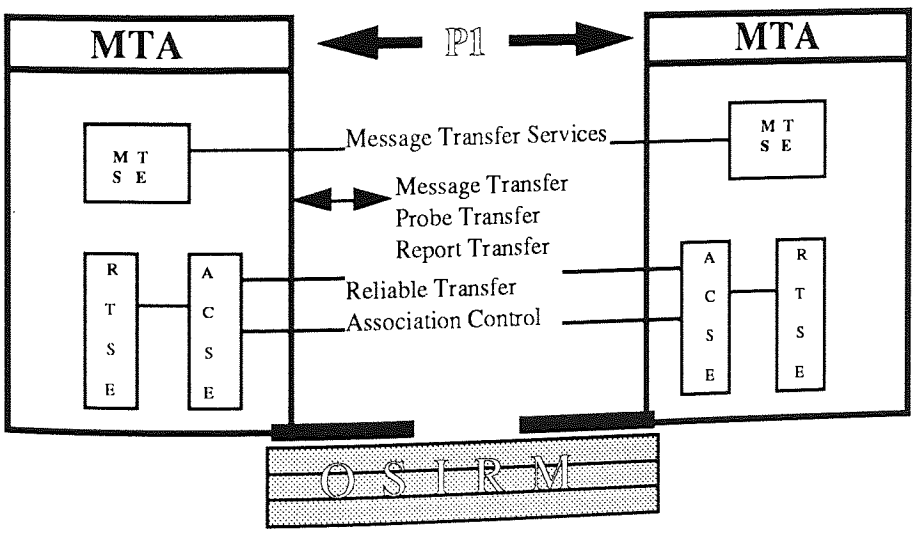


Figure A 2.3 Structure of the P1 protocol.

A 2.2 List of Suplementary Services (Transfer System & IPMS).

Basic	Access Management Content Confidentiality Content Type Indication Converted Indication Cross referencing Indication Delivery Time Stamp Indication IP-message Identification Language Indication Multi-part Body Message Identification Non-Delivery Notification Original Encoded Information Types Primary and Copy Recipients Indication Registered Information Type Stored Message Alert User/UA Capabilities Registration etc.
Submission and Delivery	Additional Physical Rendition Alternate Recipient Allowed Deferred Delivery Deferred Delivery Cancellation Delivery Notification Disclosure of Other Recipients Express Mail Service DL expansion Prohibited Latest Delivery Designation Multi-Destination Delivery Physical Delivery Notification by MHS Physical Delivery Notification by PDS Prevention of Non-Delivery Notification Special Delivery Return of Contents etc.
Conversion	Conversion Prohibition Conversion Prohibition in case of loss Information Explicit Conversion Implicit Conversion
Query	Probe Probe Origin Authentication
Status and Information	Alternate Recipient Assignment Hold for Delivery Type Body Use of Distribution List Stored Message Listing etc.

For complete Services please refer CCITT blue book VOLUME VIII - FASCICLE VIII.7. Data Communication Networks, Message Hnadling Systems Recommendations X.400-X.420.

A 2.3 An Example of Protocol Extension Macros.

```
HEADING EXTENSION MACROS ::=
BEGIN
        TYPE   NOTATION ::= "VALUE" type | empty
        VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
END

ExtensionsField ::= SEQUENCE {
        type [0] EXTENSION
        criticality [1] Criticality
        value [2] ANY DEFINED BY type }

criticality ::= BITSTRING {
        for-submission (0),
        for-transfer (1),
        for-delivery (2) }
```

The type defines the type of the extension (the EXTENSION macros defined in X.420 (88) in details) and value contains the corresponding value. Since the presence or absence of an extension may enable or prevent certain operations, the entry criticality is used to indicate the criticality or non-criticality of the corresponding extension for the operation of message submission, transfer or delivery.

The types of body parts that may appear in the body of an IPM may includes text, voice, videotex, encrypted, mixed-mode and teletex (a complete body part choice is listed in X.420 (88) etc. numbering to sixteen types.

A 2.3.1 List of X.420 fields

The Interpersonal message (IPM) is number of primary class of information object, consists of the following:

Heading fields component types,

Heading fields and

Body part types.

Heading fields component types are: **IPM identifier, recipient specifier** and **O/R descriptor** and defined as below:

IPMIdentifier ::= [APPLICATION 11] SET {

user		OR address OPTIONAL,
user-relative-identifier		LocalIPMIdentifier }

Recipient specifier ::= SET {

recipient	[0]	OR Descriptor,
notification-requests	[1]	Notification Requests DEFAULT {},
reply-requests	[2]	BOOLEAN DEFAULT FALSE }

O/R Descriptor ::= SET {

fomal-name		ORName OPTIONAL,
free-form-name	[0]	FreeFormName OPTIONAL,
telephone-number	[1]	TelephoneNumber OPTIONAL }

The fields that appear in the heading of an IPM are defined and described below:

Heading ::= SET {

this-IMP		ThisIMPField,
originator	[0]	OriginatorField OPTIONAL,
authorizing-users	[1]	AuthorizingUserField OPTIONAL,
primary-recipients	[2]	PromaryRecipientsField DEFAULT {},
copy-recipients	[3]	CopyRecipientsField DEFAULT {},
blind-copy-recipients	[4]	BlindCopyRecipientField OPTIONAL,
replied-to-IPM	[5]	RepliedToIPMField OPTIONAL,
obsoleted-IPMs	[6]	ObsoletedIPMField DEFAULT {},
related-IPMs	[7]	RelatedIPMsField DEFAULT {},
subject	[8]	EXPLICIT SubjectField OPTIONAL,
expiry-time	[9]	ExpiryTimeField OPTIONAL,
reply-time	[10]	ReplyTimeField OPTIONAL,
reply-recipients	[11]	ReplyRecipientsField OPTIONAL,
importance	[12]	ImportanceField DEFAULT normal,
sensitivity	[13]	SensivityField OPTIONAL,
auto-forwarded	[14]	AutoForwardedField DEFAULT FALSE,
extensions	**[15]**	**ExtensionsField DEFAULT** {}

The types of body that are appear in the body of an IPM are defined and described below:

```
BodyPart ::= CHOICE {
        ia5-text                [0]     IA5TextBodyPart,
        voice                   [1]     VoiceBodyPart,
        g3-facsimile            [2]     G3FacsimileBodyPart,
        g4-class1               [3]     G4Class1BodyPart,
        teletex                 [4]     TeletexBodyPart,
        videotex                [5]     VideotexBodyPart,
        encrypted               [8]     EncryptedBodyPart,
        message                 [9]     MessageBodyPart,
        mixed-mode              [11]    MixedModeBodyPart,
        bilaterally-defined     [14]    BilaterallyDefinedBodyPart,
        nationally-defined      [7]     NationallyDefinedBodyPart,
        externally-defined      [15]    ExternallyDefinedBodyPart,
```

Note: i) An O/R descriptor is an information item that identifies a user or Distribution List (DL).

ii) Some fields have components and thus are composite, rather than indivisible. A field component is called a **sub-field**.

iii) The body part of some the type may have two components that is parameters and data.

For detailed information please see section 7 (X.420), CCITT Recommendations X.400-X.420 '*Data Communication Networks: Message Handling System*' VOLUME VIII .

A 2.4 Electronic Data Interchange Messaging System (EDIMS)

The electronic data interchange messaging (EDIM) system is a service in parallel to the interpersonal messaging system. The system is can also be seen in parallel to the services to be recommended in this research. In this context it is felt necessary to describe the EDIM system in brief.

EDI (Electronic Document Interchange) is the electronic interchange of documents in all possible areas of business. The aim of EDI is to replace the numerous paper documents used in the course of business by electronic forms. Such documents include orders, delivery notes, consignment notes, customs declarations, receipts and invoices. However, to ensure the interoperability of EDI systems from different manufacturers, international standardization of the electronic representation of all possible document types is under way. The standardization is collectively denoted by EDIFACT (Electronic Document Interchange for Finance, Administration Commerce and Transport) by the United Nations Economic Commission for Europe, ISO and other standardization committees.

The importance of EDIFACT here is that it is considered to be an application of X.400 like P2/IPMS. The few EDIFACT implementations available follow a pragmatic route under X.400 and generate a P2 formatted content, the body of which contains the EDIFACT message to be transmitted. In this case the message switching nodes corresponding to the MTAs are called Clearing Centres (CC) and Terminals correspond to the UAs (Plattner et. al., 91).

Electronic Data Interchange is an application using the X.400 service as a user agent service, much like that defined for the InterPersonal Messaging (P2) service. CCITT Study Group VII has defined a new content type and protocol for EDI (for EDIFACT), currently known as Pedi (protocol for electronic data interchange) and reflected in CCITT draft Recommendation X.edi1 and X.edi2. Figure A 2.4 provides an overview of the EDIMS.

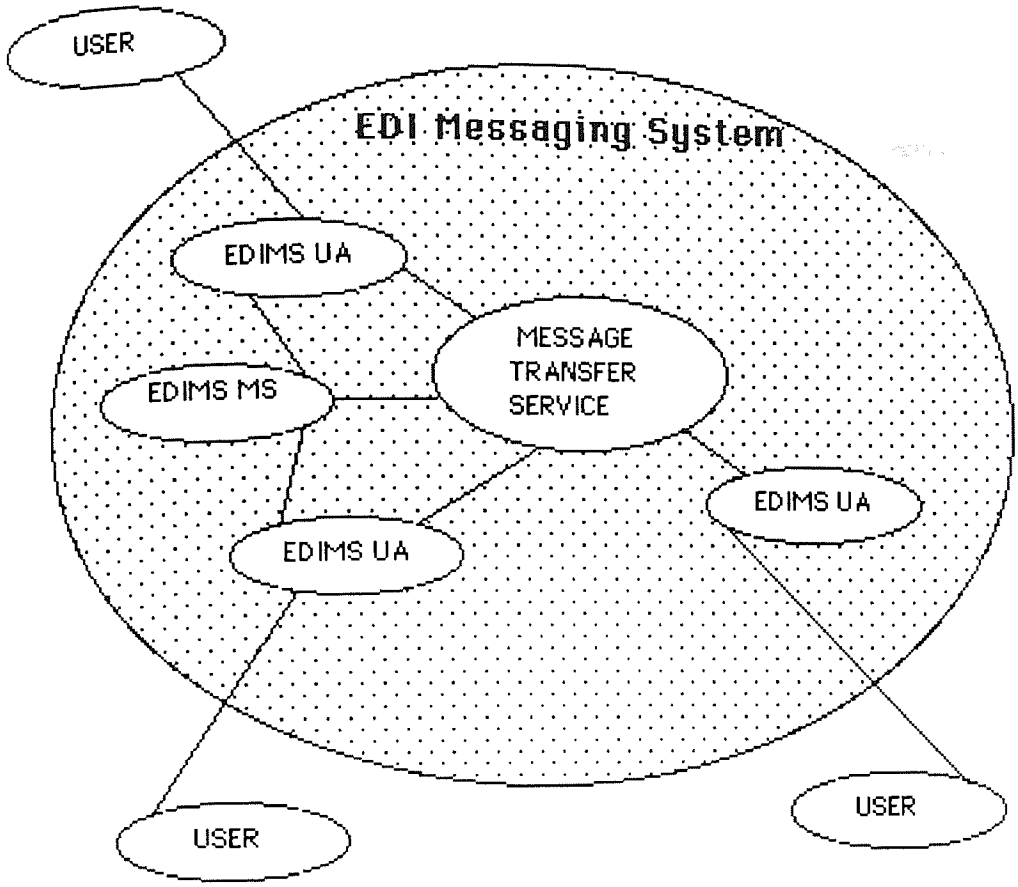
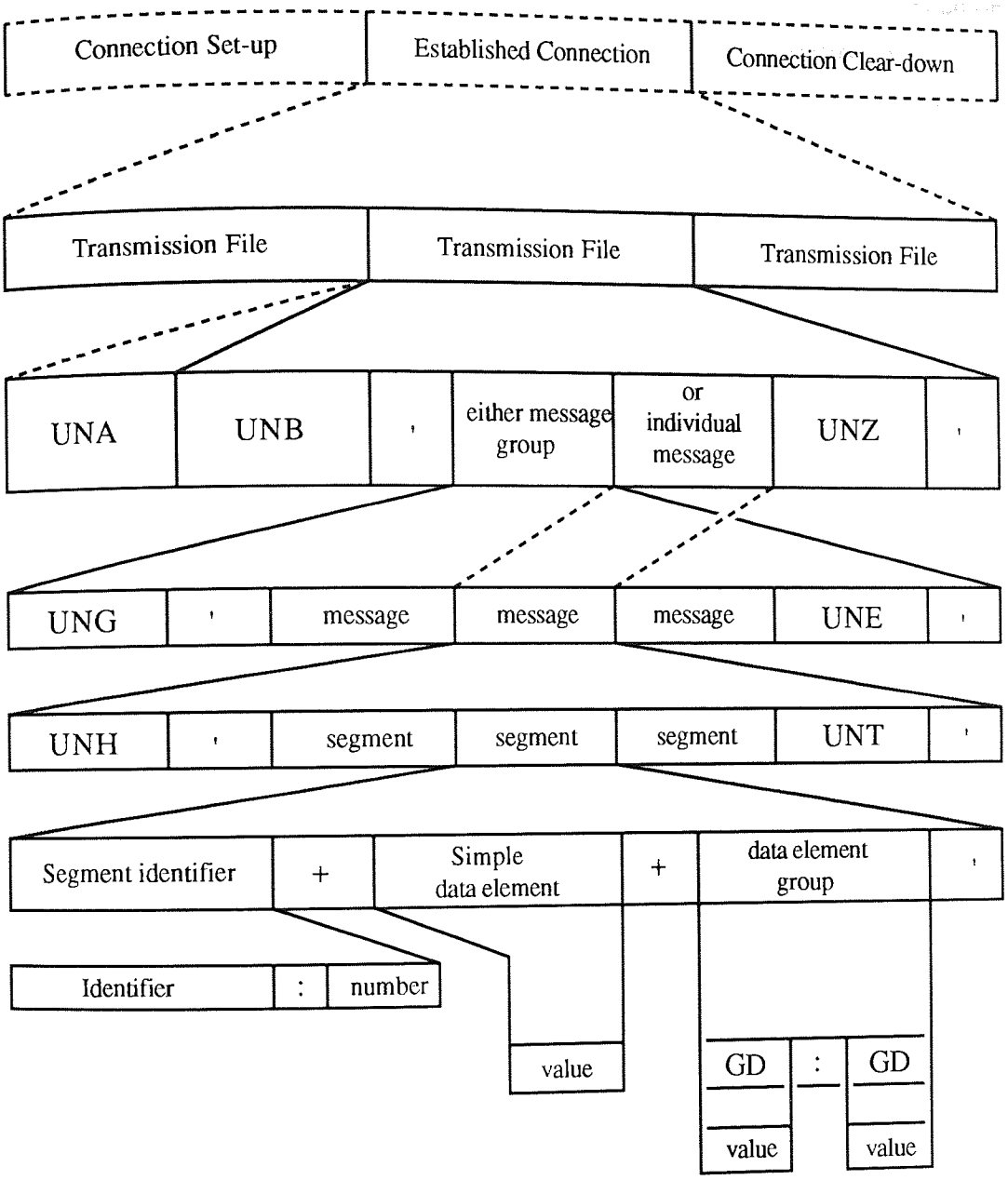


Figure A 2.4 Simple Model of the EDI Messaging System.

The EDI/EDIFACT system determines which specific EDI application is to be invoked by examination of the EDI header. For example, in the EDIFACT context, information about the entire EDI interchange is carried in an UNB[1] header. UNB is the header segment of the transfer file; it contains a statement on the standard and which version of it is used. The Pedi consists of a heading and a body, which in turn consists of a number of body parts. The body contains only one EDI message (or 'interchange', in EDIFACT terminology) or one forwarded EDI message. Other body parts may exist and may be used to contain other information, such as an explanatory note of text accompanying a purchase order. The Pedi header is a combination of P2 fields, fields from the EDI interchange header (UNB in EDIFACT), and some new fields specifically defined for Pedi. The structure of a EDIFACT transfer file is shown in Figure A 2.5.

[1]header segment of the transfer file; contains a statement on the standard and which version of it is used.

Figure A 2.5 Structure of EDIFACT file.



--------denotes alternative to _____

GD denotes group data element

For more information please refer to X.400 Message Handling, Standard, Interworking, Application, By Plattner et. al., 1991 Edition.

A 2.5 X.400 Implementation as PP

In previous sections the X.400 recommendations are described in brief. Implementations of the X.400 protocols are now available widely. It is not necessary to discuss in detail the implementation of X.400. However, in this section the 'PP' X.400 implementation is described briefly as the prototype was developed on this implementation.

The use of the PEPY ASN.1 compiler from ISODE (ISO development environment) package (Kille, 91) has made the mapping of the PP message transfer service onto OSI a relative mechanical operation. This has provided the underlying OSI model required for X.400 and QMGR ROS (queue manager remote operation service). The PEPY tool has been used by 'PP' for implementation to automatically generate the X.400 ASN.1 and the ROSY and POSY tools have been used by 'PP' for implementaion to generate the QMGR protocols.

The ISODE is a key component of the PP implementation (Kille, 91). ISODE is an implementation of the OSI upper layers. It contains the layer services of session and presentation. These are common services: the ASCE, ROSE and RTSE, application services; the FTAM and VTP and ASN.1 handling tools. The ISODE also operates over a TCP/IP network. This has enabled OSI applications, including PP and QUIPU[2] to be deployed ahead of provision of a global OSI network service. The ISODE creates an implementation environment for PP. ISODE is widely available, it is in public domain, which puts PP effectively in the public domain.

PP is a message transfer agent oriented towards support of the X.400 series recommendation. PP is implemented in C to be portable to a wide range of UNIX and Unix like operating system. The major goals of PP are given below (Kille, 91):

(i) Interface provision for message submission and delivery to support wide range of user agents,
(ii) Support of all the message transfer service elements i.e. X.400 (84 and 88),
(iii) Support to facilitate the handling of multimedia messages,
(iv) Scheduling of message delivery to optimise local and communication resources,
(v) Provision of a range of management , authorisation and monitoring facilities,
(vi) Reformation between body part types in general manner,
(vii) Support of multiple address format,
(viii) Use of OSI directory services,

[2]QUIPU is the X.500 (directory services) implementation.

(ix) Support of message protocol conversion in an integrated form,
(x) Access to message transfer services, other than IPMS and
(xi) Support of other services not provided in X.400.

A 2.5.1 Overview and Structure of PP

The PP MTA makes use of UNIX processes to provide the modularity which is fundamental to PP. The aim of a process in PP is to perform a single function, which allows flexible construction of complex processing and minimising side effects (Kille, 91). PP has a single queue which increases robustness, facilitates uniform handling of errors and simplifies the management. The structure consists of submit, queue manager, inbound and outbound channels, Queue and UA processes.

The first key process in PP is submit. Submit takes all incoming messages and places it in the queue. Submit performs all calculations to process the message and determine how it will be delivered. Early checking also ensures the detection of errors at the earliest point, and that later processes can be implemented in a more as straightforward fashion. The initial checking also includes the management and address verification. A message is received by submit either from a user agent or a protocol server called inbound channel.

The second key process is the QMGR (queue manager) which schedules all operations within PP. The QMGR holds the full queue status in memory, and controls the processes which perform the work. The major components controlled by the QMGR are channels, for the functions either of delivering a message (locally or using X.400 P1 protocol) or carrying out a format conversion (i.e. manipulation within the queue). For each MTA, there is a single queue and a single QMGR. There may be multiple instances of the submit processes acting at any one time as there may be for channel and user agent processes. The PP process structure is illustrated in figure A 2.6. There are two critical processes with complex functionality. They are the submit process and queue manager process (QMGR).

For each message in the queue there are two components to process the message; a file containing the control information and a UNIX directory containing the message as it arrives together with any associated delivery notifications.The queue encoding format is designed not to be tied to a specific protocol. It supports a wide range of protocols.

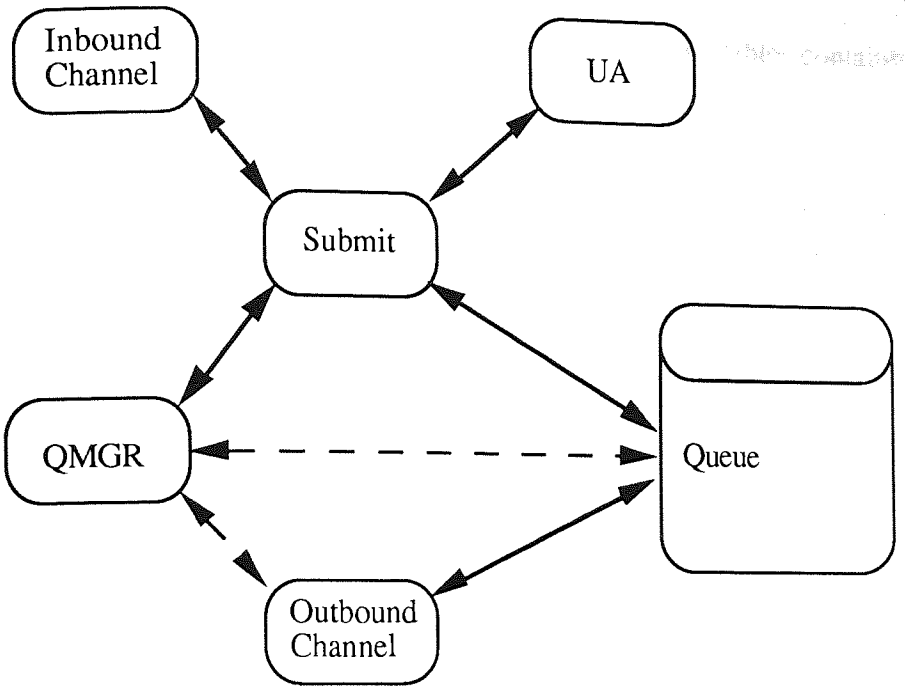


Figure A 2.6 PP Process Structure.

A 2.5.2 Specific Channel Requirement

The term channel is used in PP to describe a number of somewhat different components. In essence, each channel takes a message as an input, and then provides a different output. Channels may put a message in a queue, remove a message from a queue or transfer a message from a queue. Following channels in PP have external dependencies/supports:

(i) SMTP: The Simple Mail Transfer Protocol channel requires the use of TCP/IP.

(ii) JNT Mail: This is integrated with the UNIX-NIFTP package, which usually requires X.25 access.

(iii) X.400: This uses the ISODE lower layers (both 1984 and 1988 variants). For most services X.25, CONS or CLNP is required to run this channel, although it can be run over TCP/IP using RFC 1006 (Kille, 91).

(iv) UUCP: This channel requires a working UUCP installed on the Sunlink DNI (DecNet Interface Version 7.0 or later) package.

A 2.5.3 PP tables and their Purpose

These are five basic tables used by PP; the aliases table, the users table, the domain table, the channel table and the or table. There are also a set of tables containing authorisation information and a family of tables that provide mapping between X.400 or names and

RFC[3] 822 domain names. Specific channels may also have their own tables containing the information they require.

When a message arrives, the submit process uses the domain, or, users, aliases and channel tables to identify delivery routes for the addresses in the message. The RFC's are used to convert between X.400 style addresses and RFC style addresses and vice versa. The address arrives in one particular form either an X.400 style or an RFC 822 style address. This address is normalised via the appropriate table: the or table for X.400 style addresses, the domain table for RFC 822 style addresses. This address may be unrecognised or remote site or local. In the case of remote address, the remote site is accessed via the channel table. If the address is local than it is first looked in the aliases table and if not found then the routing information for the local address is accessed via the users table.

PPs first service operation has been to provide X.400 gatewaying services for the UK Academic Community. In providing these services, the PP system has interworked with about 40 other different implementations of P1 protocol (Kille, 91) demonstrating its interoperability. PP is being taken by sites around the world, particularly for use as a gateway or conversion service. The examples of these tables are given in below.

```
############ extract of 'aliases' table ###################
#
mail-group-request:alias Irene.Hassell
postmaster:alias Irene.Hassell
pp:alias postmaster
#
a.dacruz:synonym Alina.DaCruz
alina:synontm Alina.DaCruz
#
jpo:alias jpo@uk.ac.aston.cs 822
#
pc:alias "/I=P/S=Cowen/0=xtel/services/ADMD= /c=gb/" x400
#
```

Table A 2.1 Example of Alias Table

[3] Request For Comments.

```
# top level entries
edu:mta = 1:*
mil:mta = 1:*
pt:mta = 1:*
# local entries
cs.ucl.ac.uk:mta =
cs.ucl.ac:synonym  cs.ucl.ac.uk
cs.ucl:synonym cs.ucl.ac.uk
cs.aston.ac.uk:mta  cs.aston.ac.uk
# note the above is equivelent to
# cs.aston.ac.uk:mta =
cs.aston.ac:synonym  cs.aston.ac.uk
cs.aston:synonym cs.aston.ac.uk
cs:synonym  cs.aston.ac.uk
#
# NRS derived entries
alass.aston.ac.uk:mta alass.aston.ac.uk
alass.aston.ac:synonym alass.aston.ac.uk
alass.aston:synonym alass.aston.ac.uk
alass.aston.ac.uk:mta =
alass.aston.ac:synonym alass.aston.ac.uk
alass.aston:synonym alass.aston.ac.uk
clan.aston.ac.uk:mta clan.aston.ac.uk
clan.aston.ac:synonym clan.aston.ac.uk
clan.aston:synonym clan.aston.ac.uk
clan.aston.ac.uk:mta =
clan.aston.ac:synonym clan.aston.ac.uk
clan.aston:synonym clan.aston.ac.uk
```

Table A 2.3 Example of Domain Table

```
#
# extract of 'or' table
#
#       -*** gb ***-
#
C$GB:valid
C$UK:synonym C$GB
ADMD$GOLD 400.C$GB:valid
PRDM$UK\AC.ADMD$GOLD 400.C$GB:valid
#
#
#     -********* sites connecting via gold 400 ADMD *****-
#
PRMD$DIGITAL.ADMD$GOLD 400.C$GB:mta bt-gold
PRMD$TELECOM GOLD.ADMD$GOLD\ 400.C$GB:mta bt-gold
OU$CS.O$UCL.PRMD$UK\AC.ADMD$GOLD\ 400.C$GB:local
#
#
#                -********* france ********-
#
C$FR:valid
ADMD$PTT.C$FR:mta emu-france
ADMD$ATLAS.C$FR:mta emu-france
PRMD$BULLMTS.ADMD$ATLAS.C$FR:mta MYBULL
PRMD$DERIEUX.ADMD$ATLAS.C$FR:mta persona
#
```

Table A 2.4 Example of Or Table

```
#
#
a.cs.uiuc.edu:a.cs.uiuc.edu(smtp)
#
bsec.abcy.uwist.ac.uk:bsec.abcy.uwist.ac.uk(gb-janet)
bt-gold:bt-gold(x400out84)
bte:bte(x400out84)
abcl.co.uk:ukc.ac.uk(gb-janet), ukc.ac.uk(gb-pss)
btvax.ulster.ac.uk:btvax.ulster.ac.uk(gb-janet)
bull:bull(x400out84)
bunny.ulcc.ac.uk:bunny.ulcc.ac.uk(gb-janet)
#
emu-france:emu-france(x400out84)
#
torch.co.uk:ukc.ac.uk(gb-janet), stl.stc.co.uk(gb-pss)
# These are resolved directly by the DNS - no relay
edu:(smtp)
gov:(smtp)
com:(smtp)
us:(smtp)
mil:(smtp)
org:(smtp)
```

Table A 2.5 Example of Channel Table

```
################## extract of 'users' table ################
#
Alina.DaCruz:822-local vs2.cs.ucl.ac.uk
Steve.Kille:822-local pyr1.cs.ucl.ac.uk
John.Tayler:822-local vs2.cs.ucl.ac.uk, slocal vs2.cs.ucl.ac.uk
Postie.Pat:822-local vs2.cs.ucl.ac.uk
x400-users:list
warning:shell
```

Table A 2.6 Example of Users Table

A 2.6 Description of Make Variables (Installing PP).

Having done the normalisation of the address, the routing information for that address the address is converted into other style address (X.400 to RFC 822 or vice versa). The installation software requires a number of make variables to be set. The variables come in two classes. The first variables are truly site specific. The second variables are those which can probably be left as the defaults. A list of such variables and their description in brief is given below.

Variable	Example	Description
TAILOR	/usr/lib/pp/tailor	Location of the pp tailor file
CMDDIR	/usr/lib/pp	Directory for pp basic commands
CHANDIR	/usr/lib/pp/chans	Directory for channel programs
FORMDIR	/usr/lib/pp/format	Directory for the simple formatting channels
TOOLDIR	/usr/lib/pp/tools	Directory for miscellaneous shell scriots and debugging tools
LOGDIR	/usr/lib/pp/logs	Directory for pp logs
QUEDIR	/usr/spool/pp	Location of pp queue directory
TBLDIR	/usr/lib/pp/tables	The directory containing the table files
LIBSYS	-lisode	External libraries required - must include ISODE
LIBRESOLV	-lresolv	Resolver library for bind (empty if name server not required
USRBINDIR	/usr/local/bin	Directory for user binaries
OPTIONALCHANS	list 822-local smtp	List of optional channels required
OPTIONALFILTERS		List of optional filters
MANDIR	/usr/local/man	Directory to place manual pages under
MANOPTS	-bsd42	Manual page installation options
NIFTPSRC	/usr/src/niftp	Directory containing the source to unix-nift package (only required if you wish to run grey book mail).
INFTPINTERFACE	sun	Niftp interface being used (restriction as above)

Table A 2.7 Make Variables that should be Set at Site

Variable	Default	Description
CC	cc	The C compiler to use
CCOPTIONS	-0	C compiler options
LIBCCOPTIONS	-0	C compiler options specific to library files
OLDCC	cc	Other cc if gcc is used
LDOPTIONS	-s	Loaded options
PEPY	pepy	The *pepy* ASN.1 parser/compiler
POSY	posy	The *posy* stub generator
ROSY	rosy	The *rosy* remote operations stub generator
PEPSY	pepsy	The *pepsy* program
X11	true	Does the system have X11?
LIBX	-1Xaw -1Xmu -1X11	The X11 libraries - may need -1Xext for X11R4
LINT	lint	The *lint* (1) command
LINTFLAGS	-hbuz	lint flags
PGMPROT	755	Program protection mode
PPUSER	pp	The user id PP will run as
ROOTUSER	root	The user id for privileged
CHOWN	chown	The *chown* (8) command
CHMOD	chmod	The *chmod* (1) command
BACKUP	cp	A command to back up the old binary
INSTALL	cp	A command to install a new binary

Table A 2.8 Make Variables that can be Defaulted

More details please see 'Implementing X.400 and X.500: The PP and QUIPU Systems.

A 2.7 The OSI Directory (X.500) and Specifications

Each organisation and vendor has developed a unique and proprietary approach to the design and implementation of Directories. In the spirit of OSI, the purpose of the X.500 Directory is to provide a set of standards to govern the use of Directories.

The X.500 series of recommendations describes the operations of the Directory. It is designed to support and facilitate the communication of information between systems about objects such as data, applications, hardware, people, files, distribution lists and practically anything else that the organisation deems worthy of tracking for management purposes.

X.500 recommendations places no requirement on the nature of the information stored in the Directory (i.e. specifications for the Directory). The X.500 recommendations and their applications for the Directory Services are given in Table A 2.7.

CCITT / ISO	Applications
X.500 / 9594-1	Overview of Concepts, Models and Services
X.501 / 9594/2	The Directory Models
X.509 / 9594/8	Authentication Framework
X.511 / 9594/3	Abstract Service Definition
X.518 / 9594/4	Procedures of Dostrbited Opreations
X.519 / 9594/5	Protocol Specifications
X.520 / 9594/6	Selected Attribute Type
X.521 9594/7	Selected Object Classes

Table A 2.9 CCITT/ISO Specifications for Directories.

(For complete information please refer to CCITT blue book VOLUME VIII - FASCICLE VIII.8, Data Communication Networks, Directory Recommendations X.500-X.521.)

A 2.7.1 Features of Directory Services

The communication system consist of a large number of components of real objects including people, organisations, computers, processes, file systems and electronic mailboxes. The information relating to a real object is called a logical object (or just object) and its storage is called an entry. So that real objects in distributed applications may be unambiguously identified, they are given names. A real object may have several names. They should be user friendly.

In order to generate a communication relationship between real objects there is a requirement for addresses. An address denotes the position of a real object with respect to the system architecture. Addresses uniquely identify the real objects. The Directory Services assigns a set of values to the name of a real object. Such values include not only the address but also all valuable information in the form of text, speech, images etc.

The Directory Services includes operations to interrogate and alter the Directory. Retrieve operations may be divided into two classes (Black, 91). White pages queries provide the information stored for one or more given names. Yellow pages queries supply the names of the objects that corresponds in the criteria defined in the query. Operations involve the insertion and removal of entries of their components.

The information held in the Directory is known as the Directory Information Base (DIB). The entries in the DIB are arranged in a tree structure called Directory Information Tree (DIT).The DIB is accessed by the Directory User Agent (DUA), which is considered to be an application process. The components of the Directory system are shown in Figure A2.6.

The services are offered to users by the Directory system via a directory user agent DUA. If the DUA of the user and the Directory Service Agents (DSA) which represents the directory systems are located in different real open systems, their communication is defined by the Directory Access Protocol (DAP). When two communicating DSAs are located in different real open system their communication is defined by the Directory System Protocol (DSP) (Plattner et. al., 91).

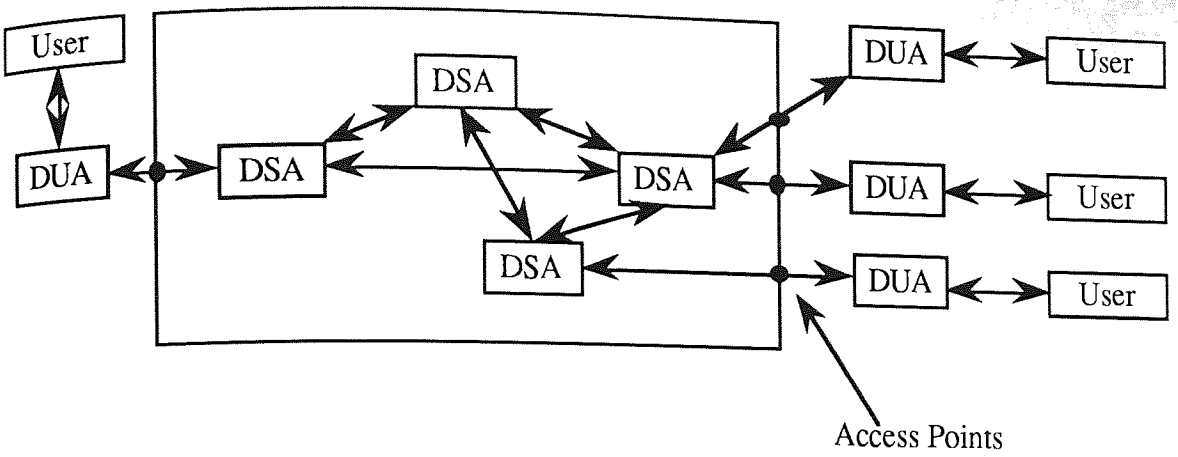


Figure A 2.7 Components of Directory System.

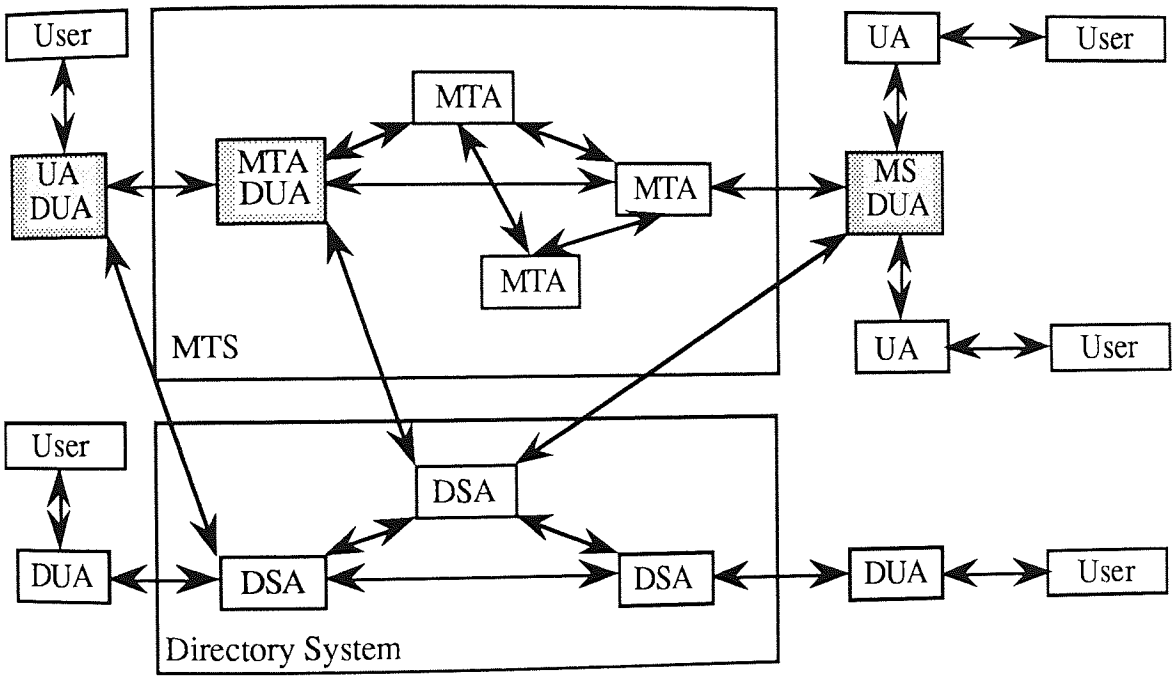


Figure A 2.8 Functional Model of the Relationship between X.500 and X.400.

Appendix 3

Characteristics of GCSA and GCUA

A 3.1 Capabilities and Characteristics of GCSA and GCUA.

Group communication service agents (GCSA) and Group communication user agents (GCUA) which are described in AMIGO report have the following capabilities and characteristics:

- A GCUA is an entry point for GCS which is reachable via a group communication port.
- A GCSA is able to co-operate with other GCSAs to provide the required services via a GC system protocol.
- In addition to co-operation, a GCSA has to co-operate with the following:
  a) the Directory Services (DS),
  b) the Archive System (AS), (AMIGO Multi-User Storage System for piloting),
  c) the message handling system (MHS '84 and 88' with message store (MS), (MHS 1984 with AMIGO-DLs for piloting),
  d) the management system (MGMS), to be based on the emerging ISO/ OSI common management information system.

A 3.2 GCS Interworking with Supporting Systems.

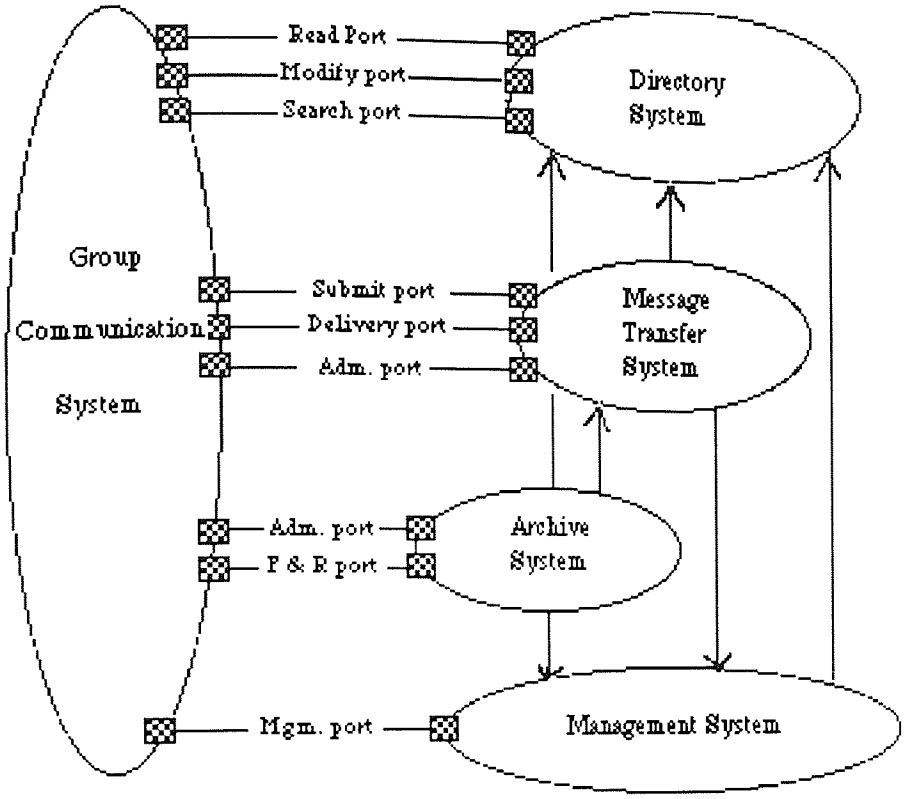


Figure A 3.1Group Communication System interworking with Supporting Systems.

For more information please see AMIGO report chapter 3: 'Group Communication Architecture'.

Appendix 4

CCITT & GRACE Operations

A 4.1 List of CCITT operations

The group communication system defines basic operation as abstract service definition. The list is in addition to the operations required to manipulates the group editing objects.

create(DN, type, attributes) --> status

Creates the object with the given distinguished name and attributes. The object is created in its distinguished domain determined from its name. It cam only subsequently be removedfrom this domain by being deleted. The object type identifies the class of object.

delete(DN) -> status

Deletes the specified object. This removes it from all other objects to which it is linked. It is not possible to delete a domain which is the distinguished naming domain for some other object.

read(DN, attribute type list) -> DN, attributes, status

Provides the mechanism for retrieving information about a single object. The operation returns the distinguished name of the object followed by the value of tthe specified attributes. It is possible to request the values of "all" attributes in the attribute type list argument.

modify(DN, modification list) -> status

Supports the modification of attributes belonging to a single object. The operation locates the named object and adds or deletes attributes as specified in the modification list. A single operation might combine the addition or deletion of several attributes. Adding an attribute involves specifying a type and a set of new values. Deleting involves specifying a type and a set of old values (the type alone means delete all values). The name of an object can not be modified with this opeartion.

link(DN1, DN2, link-type) -> status

Establishes relationship of given type between objects.

unlink(DN1, DN2, link-type) -> status

Removes relationship between two given objects. An explicit unlink between two objects will cancel a previously given link, and will also cancel existing or forthcoming derived links of the given type.

linked-to(DN, link-type, explit) -> DN list

The operation returns the names of all objects which have the specified link to the named object. If explicit is true, only direct links are found. If explicit is false, also derivrd links are found.

linked-by(DN, link-type, explit) -> DN list

This operation returns the names of all objects to which the named object has the specified link. If explicit is true, only direct links are found. If explicit is false, also derivrd links are found.

search(clusterDN, domainlist, filter, target attributes) -> DNs and attributes

This operation is used to search for the objects in a cluster or domain which match the specified filter (as in the Directory). The values of the target attributes are returned. It is possible to request the values of all attributes. If both a cluster and a domain are specified then the contents of the cluster are searched relative to the domains (i.e. only objects in the cluster and atleast one of the domain are examined).This supports the above requirement for 'views' where the results of searching a cluster may vary depending on the domain where searching is constrained.

A 4.2 List of 'GRACE' operations

The following operations are defined in the project "USENET news modelled as a GRACE activity". The operations are listed because their functionalities are similar to some of the operations require in group editing.

initialise	create the object and role to start the activity
register organisation	to register a new organisation in this domain
de-register organisation	de-register the organisation from this domain
remove item	remove old.expired items from the domain
follow-up item	respond to a specific news item
accept item	accept the item for the news group

Appendix 5

Program Listing

The program listing is devided into four part. That is part 'A', part 'B', and part 'C'. The listing description is given below:

i) **A ---> Activity Generator,**
ii) **B ---> Activity Responder**, and
iii) **C ---> Activity Monitor.**

A. ACTIVITY GENERATOR

**********MAIN **********

```
#! /bin/sh
#*************
continued="y"
service=

until [ "$continued" = "n" -o "$continued" = "N" ]
do
clear
cat < menutxt
echo -n '----------------------> ????? '
read service

case "$service" in

Ala)   defineshell;;
Blb)   chmod 666 member.dat
      add_member
      chmod 644 member.dat;;
Clc)   chmod 666 member.dat member_name
      clear
      delete_member
      mv membern.dat member.dat
       if test -s tmpname
       then
            mv member_name member.name
            del_uname
            mv member.new member_name
            mem=`cat tmpname`
            if test "$mem" = "balab"
             then
                  mv Mail/newsloc/b.bala Mail/newsloc/oldb.bala
             fi
            if (test "$mem" != "balab")
             then
                  mv Mail/newsloc/$mem Mail/newsloc/old$mem
             fi
             rm  tmpname
           chmod 644 member.dat member_name
      else
            echo "abnormal exit"
            echo " "
      fi
      echo " ";;
Dld)   notification    ;;

Ele)   process_contrib;;

Flf)   his_stat_new 2> junk ;;

Glg)   i=1

      for eall in `cat newsletter.dat`
      do
            echo $eall > news[$i]
             i=$i+1
      done
      clear
```

```
        echo "News Letter : `cat news[1+1]`          Release Date: $eall" > news.doc
        rm news[1] news[1+1] news[1+1+1]
        echo "Main Editor: bala@uk.ac.aston.quipu" >> news.doc
       echo "-----------------------------------------------------------------
        " >> news.doc

        scan +newsloc/document
        echo " "
         echo -n "Please enter the numbers (separated by spaces) of the contributions in the order they will
appear in the New Letter ?? "
       read order
        echo " "
        for msgnum in $order
        do
              show $msgnum -form doch.form >> news.doc
            echo  "-----------------------------------" >> news.doc
        done
        more news.doc
         exit
         ;;

Qlq)    exit;;

*) echo 'Please type a,b,c,d,e,f,g or q ?? ';;

esac
echo " "
echo -n "Do you need more services (default is yes type n/N to exit) ?? "
echo " "
read continued
if [ "$continued" = "n" -o "$continued" = "N" ]
then
        exit
fi
done

*************END OF ACTIVITY GENERATOR********


*******DEFINESHELL (CREATE ACTIVITY) *********
#! /bin/sh
#*************
if test -f newsletter.dat
then
        clear
        cat <<- ENDOFMSG

Warning:-
---------
1. This system supports only a single  News Letter.

2. There is a News Letter currently being prepared, if you continue
   the current News Letter will be closed.

3. If a News Letter is closed all old information will be available as follows;
(i)   activity details          --> oldnewsletter.dat
(ii)  group member details      --> oldmember.dat
(iii) remark details            --> oldremark.dat
(iv)  user names                --> oldmember_name
(v)   newsletter folder         --> oldnewsloc
```

```
Do you want to create a New News Letter(and close old one) y or n ??

y --> To create a New News Letter.

q --> To Quit (without closing existing News Letter).

ENDOFMSG

option=
echo -n 'Enter Option y or q ---> ?? '
read option

else
        option="y"
fi

if (test $option = "q" -o $option = "Q")
then
        exit
fi

if ( test $option = "y" -o $option = "Y")
then
        echo " "
        nextoption=
       echo -n "Are you sure y or q --->  ?? "
       read nextoption
fi

case $nextoption in
Yly)    if test -f newsletter.dat
        then
              chmod 644 remark.dat member.dat member_name 2> junk
               chmod 644 Mail/newsloc 2>junk
              mv member.dat oldmember.dat 2>junk
              mv remark.dat oldremark.dat 2>junk
               mv Mail/newsloc Mail/oldnewsloc 2>junk
              mv member_name oldmember_name
              mv  newsletter.dat oldnewsletter.dat 2>junk
             chmod 644 oldremark.dat oldmember.dat 2>junk
             chmod 755 oldmember_name Mail/oldnewsloc 2>junk
        fi
        rm junk
        mkdir Mail/newsloc 2>junk
        mkdir Mail/newsloc/out 2>junk
        mkdir Mail/newsloc/document 2>junk
        mkdir Mail/newsloc/reminder 2>junk
        mkdir Mail/newsloc/request 2>junk
       basic_new_defn
      chmod 644 newsletter.dat remark.dat member.dat
        ;;

Qlq) exit;;

*)      echo " "
       echo 'Please type y or q ?? '

esac
*********END OF DIFINESHELL**********
```

```
*********ADD LINK TO OLD CONTRIBUTIONS TO NEW GR. MEMBER ********
#!/bin/csh
#*************
##GET ALL GROUP MEMBERS IN GR_LST
if -f member_name then
       echo " "
       echo "linking earlier contributions to this member."
       set gr_lst = (`cat member_name`)
  if ("`cat newmember`" != "balab") then
       finger `cat newmember` >! findadr
       awk '/Directory:/ {print $2 > "foladdr"}' findadr
       set actualfold = `cat foladdr`
       foreach member ( $gr_lst )
              pick -sequence picked +newsloc/$member >! hits
               if ! -z hits then
                     refile picked -li +$actualfold/Mail/newsloc >&! junk
               endif
       end
       ##cat newmember >> member_name
       rm foladdr hits newmember
 endif
       unset gr_lst actualfold
endif
************END OF ADD LINK ************************



************NOTIFICATIONS TO SEND FIRST MESSAGE ******
#! /bin/sh
#*************
echo " "
echo -n "Enter Notification To (username@hostname): "
read note
pick -to $note -sequence picked +newsloc/out 1> hits 2> junk
if test ! -s hits
then
       echo $note > notename
       int_header
       if test -s draft
       then
              clear
              mv draft Mail/draft
              comp -use
       fi
else
      echo  Notification has already been sent
fi
#####################################
echo " " >! member_last

for member in `cat member_name`
do
       if (test $member = "bala")
       then
              set lastm=$member
       else
              if (test $member = "balab")
              then
                     echo b.bala >> member_last

             else
```

```
                echo $member >> member_last
            fi
        fi
done
echo $lastm >> member_last
#####################################

for eachmember in `cat member_last`
do
        scan +newsloc/$eachmember 1> hit 2> junk
        if test ! -s hit
        then
            if (test $eachmember != "b.bala")
            then
                pick -to $eachmember -sequence picked +newsloc/out 1> hits 2> junk
            else
                pick -to balab@email -sequence picked +newsloc/out 1> hits 2> junk
            fi
            if test -s hits
            then
                refile -li picked +newsloc/$eachmember 2> junk
                 rm hits junk
            fi
        rm hit
        fi
done

echo " "
rm member_last

**********END OF NOTIFICATION **********



**********PROCESS CONTRIBUTION**********

#!/bin/sh
#**********
#
clear
flag='f'
rm member_last 2>junk

#####################################
echo " " > member_last
for member in `cat member_name`
do
        if test $member = "bala"
            then
              set lastm=$member
        else
            if test $member = "balab"
                    then
                echo b.bala >> member_last
            else
                echo $member >> member_last
            fi
        fi
done

echo $lastm >> member_last
```

```
#####################################
awk ' {print $1 $2 > "mreader"}' member.dat

for eachone in `cat member_last`
do
        for eachmem in `cat mreader`
        do
        if (test "$eachone@quipureader" = "$eachmem")
        then
                        flag='t'
                else
                        if (test  "$eachone@email" = "eachmem")
                        then
                                flag='t'
                        fi
        fi
        done

                if test "$flag" != "t"
                then
                        echo "          contribution from $eachone "
                        scan +newsloc/$eachone
                        echo " "
                        set flag='f'
                fi
done
#
cat ed_qry
echo -n "Enter ----------> ???? "
read query

if (test $query = "q" -o $query = "Q")
then
        exit
fi
if (test $query = "e" -o $query = "E" -o $query = "a" -o $query = "A")
then

echo -n "
Carefully type in the name of the originator of the contribution ?? "
read mem
else
        exit
fi
if (test $query = "e" -o $query = "E")
        then
                if(test -s Mail/newsloc/$mem/checkedit)
                then
                        echo "Contribution is being Updated: not available for editing"
                        exit
                fi
fi
clear
for member in `cat member_last`
do
        if test "$member" = "$mem"
        then


                echo "$mem s contribution "
```

```
                scan +newsloc/$mem 2>!hh
                echo " "
                folder +newsloc/$mem
                pick -sequence picked 1> hit 2>!hh
        fi
done

if (test -s hit)
then
        echo " "
        echo -n " Select contribution number ?? "
        read number
        rm hit
else    echo "  "
        rm hit
fi

clear

if (test $query = "a" -o $query = "A")
then
        show $number | more
        anno $number -component Version: -text ' Accepted' -nodate
        refile $number -li +newsloc/document
        scan $number -format "%(msg) %{from}" |
    while read  msg from
    do


            echo -n "To: $from" > head
                show "$number" -form head.form >> head
                echo "---------------- " >> head
                show "$number" -form body.form > bodypart
        done
        cat head bodypart > Mail/draft
        send draft
fi

if (test $query = "e" -o $query = "E")
then
        if ( test $mem != "b.bala")
        then
                rm foladdr 2> junk
                finger $mem@quipu >! findaddr
        awk '/Directory:/ {print $2 > "foladdr"}' findaddr
        echo $number > "`cat foladdr`/Mail/newsloc/checkedit"
        fi
        cd Mail/newsloc/$mem
    echo $number > checkedit
        cp $number 9998
        anno 9998 -component Version: -text 'released for edit' -nodate
        cd ../../..

        scan 9998 -format "%(msg) %{from} %{cc} %{sender}" |
    while read  msg from cc sender
    do
         echo "To: geo " > head1
                if test "$mem" != "bala"
                then

                        if test "$mem" = "b.bala"
```

```
                        then
                                echo -n "Cc: $mem@email" >> head1
                        else
                        echo -n "Cc: $mem" >> head1
                        fi
                fi
         show 9998 -form head.form >> head1
        echo "Updated-On: `date` " >> head1
        echo "---------- " >> head1
    done
        show 9998 -form body.form > bodypart
        if test "`cat hostfile`" = "sparc2gx"
        then
                cat head1 bodypart > Mail/inbox/9998
                /usr/bin/X11/xmh -fg blue -bg white -bd red -flag -display sparc2gx:0 1>&2;
                rm Mail/inbox/9998
                rm `cat foladdr`/Mail/newsloc/checkedit
                rm Mail/newsloc/$mem/checkedit
                rm Mail/newsloc/$mem/9998
                exit 2
        else
                vi bodypart
                cat head1 bodypart > Mail/draft
                echo -n "Do you wish to submit this contribution Y/N ?? "
                read ans
                if (test $ans = "y" -o $ans = "Y")
                then
                        rm Mail/newsloc/$mem/9998
                        #       refile -draft -li +newsloc/$mem
                        send draft
                    rm `cat foladdr`/Mail/newsloc/checkedit Mail/newsloc/$mem/checkedit
                     echo "the contribution has been sent"
                else
                        mv Mail/draft Mail/drafts/1000
                        rm Mail/newsloc/$mem/9998
                        clear
                        echo " "
                    rm `cat foladdr`/Mail/newsloc/checkedit Mail/newsloc/$mem/checkedit
                                echo "This draft is available in the drafts folder as number 1000"
                                echo "---> EDITCHECK has been REMOVED."
                        fi
                rm head1 bodypart
                #fi
        fi
fi

#rm head head1 bodypart1 bodypart
rm member_last
#********************END OF PROCESS CONTRIBUTION ****************



********** HIS_STAT_NEW  GENERATE HISTORY AND STATISTICS *******
#! /bin/csh
#*************
set newsdata = `cat newsletter.dat`
set memberdata = `cat member.dat`
awk ' {print $1 > "memname"}' member.dat
set memnum = `cat memname`
#####################################
set gr_lst = (`cat member_name`)
```

```
echo " " >! member_last

foreach member ( $gr_lst )

      if ($member == "bala") then
            set lastm = $member
       else
            if ($member == "balab") then
                  echo b.bala >> member_last
            else
                  echo $member >> member_last
            endif
      endif
end
echo $lastm >> member_last
#######################################

#set groupdata = `cat Mail/groupm`
awk '/geo:/ {print $2 > "geoname"}' Mail/groupm
#awk '/geo:/ {print $2 > "geoname"}' aliasses
clear
echo "Application:              $newsdata[1] "
echo "News Letter:              $newsdata[2] "
echo "Completion Date:   $newsdata[3] "
echo "Group Editing Organiser: `cat geoname`"
echo "-----------------------------------"
echo " "
echo "Members Details:-"
echo "---------------- "
awk '/author / {print $1 > "authorname"  }' member.dat
echo Authors:
echo "-------"
cat authorname
echo " "
awk '/editor / {print $1 > "editorname"  }' member.dat
awk '/coordinator / {print $1 > "coordname"  }' member.dat
echo "Sub-Editors/Coordinators:"
echo "------------------------"
cat editorname
cat coordname
echo " "
awk '/reader / {print $1 > "readername"  }' member.dat
echo Readers:
echo "-------"
cat readername
echo " "
set sp = "    "
cat authorname editorname > totname
cat totname coordname > allname
cat allname geoname > totalname
echo "Overall Contribution Statistics :-"
echo "------------------------------"
echo "Total No. of Members in the Group = $#memnum"
echo " "
echo "-------------------------------------------------------------------------------"
echo "|  Member's name    | Notification |     Total          | Number of | Accepted |"
echo "|                           |  sent yes/no | Contributions   |Suggestions|  yes/no   |"
echo "-------------------------------------------------------------------------------"
foreach member ( `cat totalname`)
      pick -to $member -sequence picked +newsloc/out >! hits
       if ! -z hits then
```

```
        echo "yes" >! notes
    else
        echo "no " >! notes
    endif
    foreach uname ( `cat member_last`)

if (((($member == "$uname@quipu") || ($uname == "b.bala")) && ($uname != "bala")) then
        pick -sequence picked +newsloc/$uname >! hit
        if ! -z hit then
            awk ' {print $1 > "mnumber"}' hit
        else
            echo "-" >! mnumber
        endif
        if ( $uname == tb1 || $uname == tb2  || $uname == tb3 ) then
            echo "$member    " >! spname
        else
            if ($uname == b.bala) then
                echo "$member  " >! spname
            else
                if $uname == bala then
                    echo "$member    " >! spname
                else
                    echo "$member" >! spname
                endif
            endif
        endif
        pick -from $uname -an -search 'Revision-Suggested-By:' -sequence picked +newsloc/request >!
shits
        if ! -z shits then
            awk ' {print $1 > "sug_numb"}' shits
        else
            echo "-" >! sug_numb
        endif
        set sp="        "
        pick -from $uname -sequence picked +newsloc/document >! dhits
        if ! -z dhits then
            echo "yes" >! docs
        else
            echo "no " >! docs
        endif
        if ($member == "$uname@quipu") then
            echo "| `cat spname`   $sp `cat notes`    $sp `cat mnumber` $sp   `cat sug_numb` $sp `cat
docs`   |"
        endif
        if (($uname == "b.bala") && ($member == "balab@email")) then
            echo "| `cat spname`   $sp `cat notes`    $sp `cat mnumber` $sp   `cat sug_numb` $sp `cat
docs`   |"
        endif
endif

    end
end
echo "------------------------------------------------------------------------"
rm dhits shits hit
rm hits totalname spname sug_numb totname allname
rm geoname authorname editorname readername memname mnumber
rm notes docs coordname

**********END 0F HIS_STAT_NEW **********
```

```
*********ADD_MEMBE.C "C" PROGRAM TO ADD NEW MEMBER********
#include "basicd2.h"

FILE *rf;
FILE *mtf;

/*****************************************************************/
/*  Function get list of group members  */
struct members
getmembers()
{       FILE *mtf;
        struct members drec;
        struct members mrec;
        char temp[32];
        char stringf[32];
        char tempcmd[90];
        char inmem[140], inrec[245], yesno='N', continued='N', junk;
        int i=0, count=0;

        while ((continued != 'Y') && (continued != 'y'))
{       printf("\n\n\n\n\n\t The following questions are to obtain the details of a group member for this
News Letter.");
        while ((yesno != 'Y') && (yesno != 'y'))
        {       strcpy(inrec," ");
        strcpy(temp," ");

        printf("\n\n\n\n\t Enter e-mail address of this group member (username@hostname): ");
                scanf("%s",inmem);

        if ((rf=fopen("newmember","w"))==NULL)
        {       printf("can't open newmember\n");
                exit(1);    }
        fprintf(rf,"%s",inmem);
        fclose(rf);

        strcpy(temp,inmem);
        strcpy(tempcmd, "mkdir Mail/newsloc/");
        i=0;
        strcpy(stringf," ");
        while (temp[i] != '\100')
        {       (stringf[i]=temp[i]);
                ++i;        }
        if (strcmp(stringf,"balab") ==0)
                strcat(tempcmd,"b.bala");
        else
                strcat(tempcmd,stringf);

        if ((mtf=fopen("member.dat","a+"))==NULL)
        {       printf("can't open member.dat \n");
                exit(1);    }
        fscanf(mtf,"%s", drec.member);
while (feof(mtf) != EOF)
        if (feof(mtf) != 0) break;
        else
        {       if (strcmp(inmem,drec.member) == 0)
        {       printf("Member already exists \n");
                exit(1);    }

                fscanf(mtf,"%s", drec.status);
                fscanf(mtf,"%s", drec.topic);
```

```
                fscanf(mtf,"%s", drec.subtopic);
                fscanf(mtf,"%s", drec.member);
                ++count;                }
        fclose(mtf);
                strcpy(mrec.member,inmem);
                strcat(inmem," ");
                strcat(inrec,inmem);

                printf("\n\n\t status of group member: ");
                scanf("%s",inmem);
                strcpy(mrec.status,inmem);
                strcat(inmem," ");
                strcat(inrec,inmem);

                printf("\n\n\t Subject area for this group member: ");
                scanf("%s",inmem);
                strcpy(mrec.topic,inmem);
                strcat(inmem," ");
                strcat(inrec,inmem);

                printf("\n\n\t Topic for this group member: ");
                scanf("%s",inmem);
                strcat(inmem,"\n");
                strcat(inrec,inmem);

                printf("\n\n Is this information correct y/n ?? ");
                junk=getchar();
                yesno=getchar();        }

        system(tempcmd);

        if ((rf=fopen("member.dat","a+"))==NULL)
        {       printf("can't open member.dat \n");
                exit(1);    }

        if ((yesno = 'Y') && (yesno = 'y'))
        {       fprintf(rf,"%s",inrec);
                yesno='N';              }
        fclose(rf);
        system("addlink 2> junk");

        if ((rf=fopen("member_name","a"))==NULL)
        {       printf("can't open member_name \n");
                exit(1);    }
        fprintf(rf," %s\n",stringf);
        fclose(rf);
        printf("\n Have you entered all group members y/n ?? ");
        junk=getchar();
        continued=getchar();            }
        return  mrec;
}
/*  end of function get list of group members */
/*****************************************************************/
main()
{ FILE *mtf;
        getmembers();
printf("\n Be sureto go back to main menu and send Notification to new members.");
}

/*  end of program
/* ***************END OF ADD MEMBER PROGRAM ********************* */
```

```
**********DEL_MEMBER.C TO DELETE MEMBER ********
#include "basicd2.h"

FILE *mtf;
FILE *rf;
FILE *mf;

/* function to delete group member */
/*****************************************************************/

/* function to delete members details */
struct members
deletemember()
{           struct members trec;
            char junk, addr[64], temp[32];
            int i=0;
      char stringf[32], oldmember[32], balab[5];

            strcpy(addr," ");
      printf("\n\n Address (username@host) of the member to be deleted: ");
            junk=getchar();
      gets(addr);
            strcpy(temp,addr);
       strcpy(stringf,"                ");
      while ((temp[i] != '\100'))
       {    (stringf[i]=temp[i]);
              ++i;   }
i=0;
while (stringf[i] != '\040')
{           (balab[i]=stringf[i]);
            i++;       }

     if ((mf=fopen("tmpname","w"))==NULL)
      {      printf("can't open tmpname \n");
             exit(1);       }

if ((mtf=fopen("member.dat","r"))==NULL)
      {      printf("can't open member.dat while delete\n");
             exit(1);       }
if ((rf=fopen("membern.dat","w"))==NULL)
      {      printf("can't open member.dat\n");
             exit(1);       }
           while (feof(mtf) != EOF)
           {           if (feof(mtf) != 0) exit(0);
      else
           {           fscanf(mtf,"%s", trec.member);
                       fscanf(mtf,"%s", trec.status);

            fscanf(mtf,"%s", trec.topic);
            fscanf(mtf,"%s", trec.subtopic);
           }
           if (strcmp(addr,trec.member) != 0)
           {
                       fprintf(rf,"%s",trec.member);
                       fprintf(rf," %s",trec.status);
           fprintf(rf," %s",trec.topic);
                       fprintf(rf," %s",trec.subtopic);
           fprintf(rf,"\n");
```

```
            }
            if (strcmp(addr,trec.member) == 0)
        {   printf("Deleted Member: ");
                    printf("%s\n",addr);
                    if (strcmp(addr,"balab@email") == 0)
                            fprintf(mf,"%s\n",balab);
                    else
                            fprintf(mf,"%s\n",stringf);    }
                    strcpy(trec.subtopic, " ");
                    strcpy(trec.topic, " ");
                    strcpy(trec.status, " ");
                    strcpy(trec.member, " ");
            }
            fclose(mf);
            fclose(mtf);
            fclose(rf);
}

/*  end of function delete members details */
/*********************************************************************/

main()
{           char reply;
            printf("\n\n Do you really want to delete a member from group y/n ??  ");
            reply=getchar();

            if ((reply=='Y') || (reply=='y'))
                    deletemember();
}

/*  end of program  */

/* ***********END OF DELETE MEMBER ***************** */



**********DEL_UNAME.C***************
#include "basicd2.h"

FILE *mf;
FILE *rf;
FILE *mtf;

/* this program deletes user name from member_name file */
/*****************************************************************/

main()
{       char stringf[32];
        char oldmember[32];


if ((mtf=fopen("tmpname","r"))==NULL)
        {       printf("can't open tmpname\n");
                exit(1);        }
              fscanf(mtf,"%s", stringf);
              fclose(mtf);

if ((mf=fopen("member.name","r"))==NULL)
        {       printf("can't open member.name\n");
                exit(1);        }
if ((rf=fopen("member.new","w"))==NULL)
```

```
        {       printf("can't open member.new\n");
                exit(1);        }

                while (feof(mf) != EOF)
                {               if (feof(mf) != 0) exit(1);
        else
                                fscanf(mf,"%s", oldmember);

                if (strcmp(stringf,oldmember) != 0)
                 {              fprintf(rf,"%s\n ",oldmember);

                strcpy(oldmember," ");          }
                }

                fclose(mf);
                fclose(rf);
}
/*************END OF DELETE USER NAME*****************/



*********BASIC_HEAD.C INITIAL HEADER FOR NOTIFICATION****
#include "hbasic.h"

FILE *mtf;
FILE *mf;
FILE *rf;

/* function to application name */
/*************************************************************************/

getname() {

                char applca[24], name[64], cdate[9];

    if ((mf=fopen("newsletter.dat","r"))==NULL)
      {       printf("can't open newsletter.dat\n");
              exit(1);        }
            else
            {               fscanf(mf,"%s",applca);
                            fscanf(mf,"%s",name);
                            fscanf(mf,"%s",cdate);
            }
    fclose(mf);

    if ((rf=fopen("draft","a"))==NULL)
      {       printf("can't open draft \n");
              exit(1);        }

      fprintf(rf,"\nPart-of-News-Letter: %s",name);
      fprintf(rf,"\nLatest-Submission-Date: ");

      fprintf(rf,"\nFcc: newsloc/out");

     fprintf(rf,"\n---------- ");
           fprintf(rf,"\n");
    fclose(rf);
}

/* program to construct header for initial message */
/*************************************************************************/
```

```
main()
        {
        struct members hrec;
        char addr[64] , junk;
        strcpy(addr, " ");

/*      printf("\n\n To: ");
        gets(addr); */

/* new text for addr from shell */
    if ((mf=fopen("notename","r"))==NULL)
      {     printf("can't open notename file \n");
            exit(1);        }
        fscanf(mf,"%s", addr);

        fclose(mf);
/* end new text */

    if ((mtf=fopen("member.dat","r"))==NULL)
      {     printf("can't open member.dat file \n");
            exit(1);        }

        fscanf(mtf,"%s", hrec.member);
        fscanf(mtf,"%s",hrec.status);
        fscanf(mtf,"%s",hrec.topic);
        fscanf(mtf,"%s",hrec.subtopic);

 while ((strcmp(addr,hrec.member) != 0))
 {      fscanf(mtf,"%s", hrec.member);
        fscanf(mtf,"%s",hrec.status);
        fscanf(mtf,"%s",hrec.topic);
        fscanf(mtf,"%s",hrec.subtopic);
        if (feof(mtf) !=0)
        {       printf("record not found \n");
                exit(1);    }       }
        fclose(mtf);
if (strcmp(addr,hrec.member) != 0)
        {       printf("record not found \n");
          exit(1);      }
   if ((rf=fopen("draft","w+"))==NULL)
      {     printf("can't open draft \n");
            exit(1);        }
        fprintf(rf,"\To: %s",hrec.member);
        fprintf(rf,"\nStatus: %s",hrec.status);
        fprintf(rf,"\nSubject: %s",hrec.topic);
        fprintf(rf,"\nTopic: %s",hrec.subtopic);
        fclose(rf);
        getname(); }
/*  end of program  */
/* ************END OF HEADER FOR NOTIFICATION**************** */


**********BASIC_NEW_DEFN.C TO CREATE NEW NEWS LETTER ****
#include "basicd2.h"

#define  THISYEAR 92

/*  function define application file  */

FILE *rf;
struct mfilentry
```

```
getdefin()
{       struct mfilentry frec;
        char junk, confirmed='n';
        char inword[64], inline[95], remark[250] ;
        printf("\f\n\n\n Please enter basic histocal details for the application. ");
        printf("\n\n\t\t `EDITING NEWS LETTERS`");
        while ((confirmed != 'y') && (confirmed != 'Y'))
        {       strcpy(inline,"Editing-News-Letters ");
                printf("\n\n\n\n\n\t Name & Issues of the News Letter: ");
                gets(inword);
                strcpy(frec.name,inword);
                strcat(inword," ");
                strcat(inline,inword);
                printf("\n\n\t Date of Release: ");
                scanf("%s",inword);
        /*      while ((chdate(inword))==0)
                {   printf("\n enter date of meeting in format DDMMYY.");
                    scanf("%s",inword);   } */
                strcpy(frec.date,inword);
                strcat(inword," ");
                strcat(inline,inword);
                junk=getchar();
                        printf("\n\n\t Remark if Any ?     ");
                gets(remark);
                printf("\n   ..?? Is this correct  ? Y/N  ");
                confirmed=getchar();
                junk=getchar();                 } /* end of while loop */

        if ((rf=fopen("remark.dat","w+"))==NULL)
        {       printf("can`nt open remark file.");
                exit(1);    }
        fprintf(rf,"%s",remark);
        fclose(rf);

        if ((mtf=fopen("newsletter.dat","w+"))==NULL)
        {       printf("can`nt open news letter file.");
                exit(1);    }
        fprintf(mtf,"%s",inline);
        fprintf(mtf,"%s","\n");
        fclose(mtf);
        return      frec;
}
 /* end of  define application function  */
/*************************************************************/

/* function to check date validity */

chdate(indate)
char *indate;
{       int idate, iday, imonth, iyear, valid = 1;
        idate = atoi(indate);   /* convert string to integer */
        iday = idate/10000;
        imonth = (idate%10000)/100;
        iyear = idate%100;
        /* printf("%i",iday); */


        if ((strlen(indate) != 6) || ((iday<1) || (iday>31))
                || ((imonth<1) || (imonth>12)) || (iyear<THISYEAR))

/*  {                           strcpy(indate,iday);
```

```
        strcat(indate,"-");

                        strcpy(indate,imonth);
        strcat(indate,"-");
                        strcpy(indate,iyear);
        printf("\n\n %s",indate);       } */
                        valid =0;
        return valid;
}
/* end of date valid function */
/*****************************************************************/


main() {
        char addmem='n';
        struct members aprec;
        struct mfilentry nrec;
        getdefin();
        printf("\n\n\t\t Want to Add Members ?? ");
        addmem=getchar();
        if ((addmem == 'y') || (addmem == 'Y'))
        system("add_member");

/*      aprec=getmembers();             */

}

/*  end of program  */

/* ************END OF BASIC_NEW_DEFN.C*************** */


*******HEADER FILE FOR ADD DELETE AND CREATE MEMBER **
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define  CENTURY 19

FILE *mf;
FILE *mtf;

struct mfilentry {
        char applica[24];
        char name[64];
        char date[7];   } ;

struct members  {
        char member[140];
        char status[25];
        char topic[40];
        char subtopic[40];    } ;

/* function to read application defination record  */
/*****************************************************************/
/*   function read definition */
struct mfilentry
readdefin()
{
        struct mfilentry drec;
        char inword[64], inline[140];
        int count=0;
```

```
        fscanf(mf,"%s", drec.applica);
        strcpy(drec.applica, "Editing News Letters");
        strcpy(inline," ");

        if (feof(mf) != 0)
        {       printf("\n can`nt open definition file.\n");
                exit(1);    }
        else
        {
                scanf(mf,"%s",inword);
                while (strcmp(inword,"**") != 0)
                {       strcat(inline,inword);
                        strcat(inline," ");
                        fscanf(mf,"%s",inword);  }
                fscanf(mf,"%s",drec.name);

                fscanf(mf,"%s",inword);
                fscanf(mf,"%s",drec.date);
        }
        return drec;
}

/*  end of function readdefin */
/*****************************************************************/
/*   function read menbers */

struct members
readmember()
{       struct members mrec;
        char inword[64], inline[140];
        fscanf(mtf,"%s", mrec.member);
        strcpy(inline, " ");

        if (feof(mtf) != 0)
        {       printf("\n normal exit \n");
                exit(1);    }
        else
        {       fscanf(mtf,"%s",inword);
                while (strcmp(inword,"**") != 0)
                {       strcat(inline,inword);
                        strcat(inline," ");
                        fscanf(mtf,"%s",inword);  }
                fscanf(mtf,"%s",mrec.status);
                fscanf(mtf,"%s",inword);
                while (strcmp(inword,"**") != 0)
                {       strcat(inline,inword);
                        strcat(inline," ");
                        fscanf(mtf,"%s",inword);  }
                fscanf(mtf,"%s",mrec.topic);
                fscanf(mtf,"%s",inword);
                while (strcmp(inword,"**") != 0)
                {       strcat(inline,inword);
                        strcat(inline," ");
                        fscanf(mtf,"%s",inword);  }
                fscanf(mtf,"%s",mrec.subtopic);
                fscanf(mtf,"%s",inword);  }
        return mrec;
}
/*  end of read member funtion   */
/**************END OF HEADER FILE BASICD2.H*************/
```

```
********HEADER FILE HBASIC.H HEADER FOR NOTIFICATION****
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define  CENTURY 19

FILE *mf;
FILE *mtf;

struct mfilentry {
        char applica[24];
        char name[64];
        char date[7];    } ;

struct members  {
        char member[140];
        char status[25];
        char topic[40];
        char subtopic[40];    } ;

/* function to read application defination record  */
/*****************************************************************/

/*   function read definition */

struct mfilentry
readdefin()
{
        struct mfilentry drec;
        char inword[64], inline[140];
        int count=0;
        fscanf(mf,"%s", drec.applica);

        if (feof(mf) != 0)
        {       printf("\n can`nt open definition file.\n");
                exit(1);    }

        else
        {       fscanf(mf,"%s",drec.name);
                fscanf(mf,"%s",drec.date);
        }
        return drec;
}
/*  end of function readdefin */
/*****************************************************************/
/*   function read menbers */
struct members
readmember()
{       struct members mrec;
        char inword[64], inline[140];
        strcpy(inline, " ");

        if (feof(mtf) != 0)
        {       printf("\n normal exit \n");
                exit(1);    }
        else
        {       fscanf(mtf,"%s", mrec.member);
                fscanf(mtf,"%s",mrec.status);
```

```
                    fscanf(mtf,"%s",mrec.topic);
                    fscanf(mtf,"%s",mrec.subtopic);          }
          return mrec;}
/*  end of read member funtion  */
/************END OF HBASIC.H*****************/


***********HEAD.FORM HEADER FILE*********
:
leftadjust,compwidth=12
ignores=msgid,message-id,received,cc,forwarded,via,updated-on,return-path,to,from,date
extras:nocomponent

**********end of head.form **********


**********BODY.FORM BODY FILE***************

leftadjust,compwidth=24
:
body:nocomponent,overflowtext=,overflowoffset=0,noleftadjust

**********end of body.form**********


**********DELIVNOTE.FORM ***********
leftadjust,compwidth=12
ignores=msgid,message-id,received,forwarded,return-path,from,date
extras:nocomponent
:
body:nocomponent

***********end of delivnote.form*********


*************DOCH.FOM TO GENERATE DOCUMENT**********
:
leftadjust,compwidth=10
Subject:
ignores=msgid,message-id,via,received,forwarded,return-path,to,date,version,status,Updated-On,Latest-
Submission-Date,Part-of-News-Letter
extras:nocomponent
:----------
body:nocomponent

*************end of doch.form ***************


**********EDIT.FORM TO EDIT CONTRIBUTION**********
:
leftadjust,compwidth=10
To:
Subject:
ignores=msgid,message-id,received,forwarded,return-path,date,version
extras:nocomponent
:----------
```

body:nocomponent

**********end of edit.form *********

**********RELEASE.FORM TO RELEASE CONTRIBUTION********
leftadjust,compwidth=12
ignores=msgid,message-id,received,forwarded,return-path,version,date
extras:nocomponent
:
body:nocomponent

**********end of release.form *********

**********MHL.FORMAT MAIN FORMAT FILE TO SHOW **********
: -
leftadjust,compwidth=12
To:
From:
Status:
Subject:
Topic:
ignores=msgid,message-id,received,date,via,return-path,forwarded
extras:nocomponent
:
body:nocomponent,overflowtext=,overflowoffset=0,noleftadjust

********** end of mhl.format **********

**********menu text file for generator ********

'Welcome to News Letter Editing System'

Please Enter the Letter for the Service you Require

a --> To Create New NewsLetter.

b --> To Add Group Members.

c --> To Delete Group Member.

d --> To Send First Notification to Group Members.

e --> To Process Contribution.

f --> To History & Statistics.

g --> To Generate News Letter Document.

q --> To Quit.

**********end of menu file **********

****** menu file (ed_qry) process contribution *******

---> a to accept contribution

---> e to edit contribution

---> q to quit
**********end of query file **********

B. ACTIVITY RESPONDER

```
**********Main script for responder ********
#!/bin/sh
#********
set continued="y"
until [ "$continued" = "n" -o "$continued" = "N" ]
do
process_edit 2>junk
echo -n "Do you need more services default is yes (to quit type n/N) ??"
read continued
if [ "$continued" = "n" -o "$continued" = "N" ]
then
        exit
fi
done
**********end of Main **********

**********process_edit  main query system *******
#!/bin/sh
#*********
#*THIS IS MAIN SCRIPT RUNS AT MEMBER'S END
#*THE SCRIPT IS MENU BASED AND NEEDS MANUAL INTERACTION OF THE MEMBER
#*CONCERNED TO WORK WITH THE NEWS LETTER.
#*THE SCRIPT DOES AUTOMATIC DELIVERY FAILURE CHECKING AND IF FOUND,
#*THE FAILED CONTRIBUTION IS RESENT TO THE GEO.
#*THE SCRIPT ALSO CHECKS REMINDER AND IF FOUND, AMESSAGE IS DISPLAYED TO
THE MEMBER.
#CHECK FOR REMINDER
check_reminder 2> junk
#CHECK FOR DELIVERY FAILURE
check_delivery 2> junk
set continued="y"
until [ "$continued" = "n" -o "$continued" = "N" ]
do

        inc -silent 2> junk
        folder +inbox 1> junk 2> junk
#*CHECKING FOR THE FIRST MESSAGE TO START THE NEWS LETTER
#*IF MESSAGE IS FOUND IT IS REFILE TO NEWS LETTER OUT FOLDER
        scan +newsloc/out 1> hits 2> junk
if [ ! -s hits ]
then
        pick +inbox -search 'Part-of-News-Letter:' -sequence picked 1> hit 2> junk
        if [ -s hit ]
        then
                refile picked +newsloc/out
        fi
        rm hit 2> junk
fi
rm hits 2>junk
#*CHECKS FOR CARBON COPY RECEIVED IF GEO EDITS THE OTHER MEMBER'
CONTRIBUTION
#*IF FOUND REFILES TO THE NEWSLETTER OUT FOLDER
pick +inbox -lb -cc $USER -an -search 'Part-of-News-Letter:' -rb -sequence picked 1> hit 2> junk

if [ -s hit ]
then
        refile picked +newsloc/out
fi
rm hit
```

```
#*CHECKS FOR NEWS LETTER FOLDERS IF EMPTY THEN EXITS AFTER DISPLAY OF A
MESSAGE.
scan +newsloc/out 1> hits 2> junk
if [ ! -s hits ]
then
        scan +newsloc 1> hit 2> junk
        if [ ! -s hit ]
        then
                        echo "EITHER"
              echo "common store is empty"
                        echo "or you are not recorded as member of the group"
                        echo "or a contribution request has not been made to you"
                        echo " "
               exit
        fi
            rm hit 2>junk
fi
rm hits 2>junk

#*DISPLAYS QUERY MENU AND WAITS FOR THE REPLY FROM THE MEMBER TO ACT ON
clear
cat queryfile
echo -n "Enter option please  ?? "
read job
echo " "
case $job in

#*CHECKS FOR WORKSTATION, IF WORKSTATION IS SPARC2GX THEN OLD DRAFTS ARE
#*OPENED IN A xmh WINDOWING ENVIRONMENT
Blb)        if  ( test -s hostfile -a "`cat hostfile`" = "sparc2gx" )
            then
                        /usr/bin/X11/xmh -fg blue -bg white -bd red -flag -initial drafts -display sparc2gx:0
1>&2;
                        exit 2
            else
                        echo " "
                        echo "quipu is to be added on spac2gx"
                        echo "please run xlinking if workstation is sparc2gx"
                        echo " "
            fi
            ;;

#*DISPLAYS OLD VERSIONS TO MAKE EDIT REQUEST FOR A CONTRIBUION
#*AND WAITS FOR CONTRIBUTION NUMBER TO ENTERED AND
#*SENDS THE REQUEST TO COMMON STORE
Clc)        if [ -s Mail/newsloc/checkedit ]
            then
            echo "Contribution is being updated: not available for editing"
                  exit
            else
                        scan last +newsloc/out -format "%(msg)" 1> msgnum
                        mnumber=`cat msgnum`
                        if test "$mnumber" -lt 2
                        then
                                    echo " "
                                    echo you can edit first time
                                    echo " "
                                    exit
                        else

                                    scan +newsloc/out
```

```
                        cat request_text
                        xrequest
                fi
        fi
        echo "there will be a communication delay before the contribution arrives"
         ;;

#*CHECKS FOR THE RELEASED CONTRIBUTION FOR NEXT UPDATE, IF FOUND THEN
CONSTRUCTS
#*THE HEADER FOR THE NEXT CONTRIBUTION AND OPENS THE BODY PART FOR UPDATE.
Dld)    folder +inbox 1> junk 2> junk
        pick -search 'Version: released for edit' -sequence picked 1>hits 2>junk
        if [ -s hits ]
        then
                echo -n "To: geo" 1> head
         show picked -form head.form >> head
        echo "Updated-On: `date` " 1> head1
                echo "Fcc: newsloc/out" >> head1
        echo "---------------- " >> head1
                cat head head1 > headpart
                show picked -form body.form 1> bodypart

#*CHECKS FOR WORKSTATION, IF WORKSTATION IS SPARC2GX THEN OLD DRAFTS ARE
#*OPENED IN A xmh WINDOWING ENVIRONMENT
                if ( test "`cat hostfile`" = "sparc2gx" )
                then
                        cat headpart bodypart 1> Mail/inbox/9999
                        echo " PLEASE SAVE THE EDITED DRAFT IF DO NOT WANT TO
SEND AND UPDATE NEXT TIME"
                        /usr/bin/X11/xmh -fg blue -bg white -bd red -flag -display sparc2gx:0 1>&2;
                        rmm 9999 +inbox
                        rmm picked
                        exit 2
                else
                        vi bodypart
                        cat headpart bodypart 1> Mail/draft
                        echo " "
#*IF THE CONTRIBUTION IS NOT SNET THEN THE SAME DRAFT CAN BE EDITED
#*WHICH IS AVAILABLE AS MAIL/DARFT IF REQUIRES OTHERWISE FRESH DRAFT WILL BE
GENERATED
                        echo -n "Do you wish to send this contribution ?? "
                read ans
                if (test $ans = "y" -o $ans = "Y")
                 then
                     send draft
                                echo " "
                                echo contribution sent successfully
                                echo " "
                                rmm picked
                        fi
                        rm bodypart head head1 headpart hits junk
                fi
        else
                echo " "
                echo no edit request made or document not received
                echo " "
        fi
        folder +inbox 1> junk 2>junk            ;;
```

```
#*DISPLAYS ALL CONTRIBUTION BY OTHER MEMBER'S FOR SUGGESTION
#*AND WAITS FOR REPLY TO SELECT THE CONTRIBUTION NUMBER AND SELECTED
CONTRIBUTION IS SENT  #*TO THE GEO AFTER ADDING REVISION-SUGGESTED-BY:
COMPONENT AS HEADER PARAMETER
Ele)    scan +newsloc
        echo " "
        echo -n "Enter the number of the contribution on which you want to make a suggestion ?? "
        read thisnum
        echo " "
        echo "Type control D when finished typing."
        echo " "
        sleep 2
        cp Mail/newsloc/$thisnum Mail/newsloc/9990
        anno 9990 -component Revision-Suggested-By: -text  "$USER@`hostname`" -nodate
        show -form suggest.form 9990 1> Mail/draft
        comp -use
        rmm 9990
        ;;

#*THIS IS C SHELL TO WORK OUT DIFFERNCES OF TWO CONTRIBUTIONS BY A MEMBER
Flf)    checkdiffer 2>! junk
         ;;

#*CREATES HEADER FOR NEWS LETTER AND FIRST DRAFT BASED UPON THE FIRST
MESSAGE SENT BY GEO
#*THE BODY PART IS PRESENTED FOR UPDATE
Ala)    folder +newsloc/out 1>! junk 2>! junk
        scan last -format "%(msg)" 1> msgnum
        mnumber=`cat msgnum`
        if test "$mnumber" -gt 1
        then
                echo " "
                echo please make edit request
                echo " "
                exit
        else
                echo -n "To: geo" 1> head
                show 1 +newsloc/out -form head.form >> head
        echo "Updated-On: `date` " >> head
                echo "Fcc: newsloc/out" >> head
                echo "---------------- " >> head
                echo "Message From geo :-" 1> bodypart
                show 1 +newsloc/out -form body.form 1>> bodypart

#*CHECKS FOR WORKSTATION, IF WORKSTATION IS SPARC2GX THEN OLD DRAFTS ARE
#*OPENED IN A xmh WINDOWING ENVIRONMENT
                if test "`cat hostfile`" = "sparc2gx"
                then
                        cat head bodypart 1> Mail/inbox/9999
                        /usr/bin/X11/xmh -fg blue -bg white -bd red -flag -display sparc2gx:0 1>&2;
                        rmm 9999 +inbox
                        exit 2
        else
                        vi bodypart
                        cat head bodypart 1> Mail/draft
                        echo -n "Do you wish to send this contribution ?? "
              read ans
               if (test $ans = "y" -o $ans = "Y")


               then
```

```
                        send draft
                        echo contribution sent successfully
                fi
        fi
                rm head bodypart junk
        fi
         ;;

Qlq)    exit;;

esac
#*CONTINUES TILL QUERY IS NOT RESPONDED n
#echo -n "Do you need more services default is yes (to quit type n/N) ??"
#read continued
#if [ "$continued" = "n" -o "$continued" = "N" ]
#then
#       exit
#fi
done

**********end of process_edit **********



*******checkdiffer to check differences of two contributions*****

#! /bin/csh
#**************
folder +newsloc >&! junk
clear
########################################
set gr_lst = (`cat member_name`)
echo " " >! member_last

foreach member ( $gr_lst )

      if ($member == "bala") then
            set lastm = $member
       else
            if ($member == "balab") then
                  echo b.bala >> member_last
             else
                  echo $member >> member_last
             endif
      endif
end
echo $lastm >> member_last
####################################
set memlist = `cat member_last`
foreach member ($memlist)
set i = 1
#clear
  if (($member != "b.bala") && ($member != "bala")) then
          pick -from $member -sequence picked >! hits
          if ! -z  hits then
                    scan picked -format "%(msg)" >! msgnum
                    set numlist = `cat msgnum`



                    foreach num ($numlist)
```

```
                                show $num -form body.form >! fil$i
                                if ($i > 1) then
                                  @ j = $i - 1
                                  diff fil$j fil$i >! thisdiff
                                  echo " "
                                  echo "Member $member --Difference of contributions $j and $i is -->"
                                  cat thisdiff | more
                                  echo "----------"
                                  #rm fil$j fil$i
                                endif
                                @ i++
                        end
                        sleep 1
                endif
                refile picked +newsloc/newstmp >&! junk
        endif
end
folder +newsloc/newstmp >! junk
refile all +newsloc
echo " "
echo " "
echo "Check this ---------->"
set i = 1
pick +newsloc/out -sequence picked >&! junk
scan picked -format "%(msg)" >! msgnum
        set numlist = `cat msgnum`
        foreach num ($numlist)
                show $num -form body.form >! fil$i
                if ($i > 1) then
                        @ j = $i - 1
                        diff fil$j fil$i >! thisdiff
                        echo " "
                        echo "Member "$user" --Difference of contributions $j and $i is -->"
                        cat thisdiff | more
                        echo "----------"
                endif
                @ i++
        end
sleep 1
folder +inbox >! junk
rm member_last

**********end of checkdiffer**********



**********check_reminder ******** for over due ********
#!/bin/csh
#**********
pick -subject 'Reminder' -an -search 'Part-of-News-Letter:' -sequence picked >! hits
if ! -z hits then
        echo " Got reminder to edit SEE in newsloc Folder"
        refile picked +newsloc
endif
rm hits

**********end of check_reminder**********



*******check_delivery to check delivery failure *********
```

```
#!/bin/csh
#************
echo "checking for delivery failure"

folder +inbox >&! /dev/null

##CHECK FOR MESSAGE AS DELIVERY FAILURE
pick -search 'Message_Type: Delivery Report' -sequence picking >! hits
if ! -z hits then
      show picking >! hit

##IF THERE IS DELIVERY FAILURE MESSAGE THEN GET ITS INITIAL DELIVERY TIME
      awk '/Date:/ {print  $2 $3 $4 $5 $6 $7 > "intialtime"}' hit
      set ctime = `cat intialtime`
      folder +newsloc/out >&! junk

##GET MESSAGE NUMBER OF ALL OUT GOING MESSAGES
     scan -format "%(msg)" >! checkfail
     set msgtime = `cat checkfail`
     foreach ftime ($ctime)
           foreach eachmsg ($msgtime)
                 show $eachmsg -noshowproc >! hit

##GET DELIVERY TIME OF EACH OF OUT GOING MESSAGE
                 awk '/^D/{print $2 $3 $4 $5 $6 $7 > "findtime"}' hit
                 set foundtime = `cat findtime`

##IF THE DELIVERY FAILURE MESSAGE TIME IS SAME AS THIS MESSAGE TIME
##THAN RESEND THIS MESSAGE
                 if ($foundtime == $ftime) then
                       show $eachmsg -form delivnote.form >! Mail/msgresend
                       send msgresend
                 endif
           end ##end of for loop
     end ##end of for loop
endif
folder +inbox >! junk

**********end of delivery failure *****************



*********suggest.form **********

leftadjust,compwidth=12
To:
From:
ignores=msgid,message-id,via,received,updated-on,return-path,date
extras:nocomponent
:
body:nocomponent
*****************end of suggets.form********
```

*********queryfile menu for responder *********

'Welcome to News Letter Editing System'

Please Enter the Letter for the Service you Require

a ---> To create first draft

b ---> To edit earlier drafts (if workstation supports X Windows)
(Presently sparc2gx only)

c ---> To select a stored contribution for editing

d ---> To edit contribution released from common store

e ---> To suggest revision to contributions by other members

f ---> To show differences between two contributions

q ---> To quit

**********end of queryfile**********

**********request_text message for edit contribution *********

Enter the number of the Contribution you wish to edit from the list above.

Then, if you wish, enter explanatory notes for recording in the activity database.
Press control D immediately if there are no notes,
otherwise press control D after you have finished entering the notes.

After you have pressed control D, the prompt what now ? will appear:
Type s to send or q -d to quit (q 'space' -d).

**********end of request_text **********

C. ACTIVITY MONITOR.

```
#!/bin/sh
#*********
clear
monitor_sys 2>junk
#!/bin/csh
#**********
#*THIS C SHELL PROVIDES BUILT-IN PROCESSOR FACILITIES AT
#*GR. EDITING ORGANISER'S END.
#*THIS SCRIPT SHOULD RUN AS BACK GROUND JOB AT PREDEFINED TIME INTERVALS
#*FOR AUTOMATIC HANDLING AND UPDATE OF THE NEWS LETTER ACITVITY ISSUES
LIKE;
#* -CHECKING OF EDIT REQUEST,
#* -RELEASE OF REQUESTED VERSION FROM COOMON STORE IF AVAILABLE FOR EDIT,
#* -CHECKING FOR NEW CONTRIBUTIONS TO BE MADE AVAILABLE FOR OTHER MEMBERS,
#* -CHECKING BFOR SUGGESTION CONTRIBUTIONS,
#* -CHECKING FOR DELIVERY FAILURE,
#* -CHECKING FOR REMINDERS AND ISSUEING REMINDERS IF REQUIRED.
# this is working shell to link message to all users.
####################################
set gr_lst = (`cat member_name`)
echo " " >! member_last

foreach member ( $gr_lst )

        if ($member == "bala") then
                set lastm = $member
        else
                if ($member == "balab") then
                        echo b.bala >> member_last
                else
                        echo $member >> member_last
                endif
        endif
end
echo $lastm >> member_last
####################################
##GET ALL GROUP MEMBERS IN GR_LST
set gr_lst = (`cat member_last`)
echo " List of Group Members: $gr_lst"
echo " "
##SET VARIABLES
set i = 1
set bdr = /isode/csam/
set mdr = /Mail/newsloc/
set odr = /Mail/newsloc
set rmr = /Mail/newsletter
##SET REMINDER TIME AS 15 ADYS FROM THE TIME OF FIRST MESSAGE
set checktime1 = 6696000
set checktime2 = 1648000
folder +inbox >&! junk
##INCORPORATE ALL NEW INCOMING MESSAGES
inc -silent >&! junk

##GET GROUP MEMBERS DIRECTORY PATH ON THEIR HOST
foreach member ( $gr_lst )
        if ($member != "b.bala") then
                finger $member@quipu >! findaddr$member
        endif
end
```

```
###*****************************************************************
echo checking to incorporate new or updated contributions, or suggestions
echo " "

foreach member ( $gr_lst )
##GET GROUP MEMBERS DIRECTORY PATH ON THEIR HOST
        pick -from $member -an -search 'Revision-Suggested-By:' -sequence pickedup +inbox >! hits
        if (! -z hits) then
###########
if ($member != "b.bala") then
                awk '/Directory:/ {print $2 > "foladdr"}' findaddr$member
                set actualfold = `cat foladdr`
                refile pickedup -li +$actualfold/$odr >&! junk
endif
############
                refile pickedup +newsloc/request >&! junk
                folder +inbox >&! junk
        endif

##PICK ALL CONTRIBUTIONS WHICH HAS HEADER COMPONENT 'VERSION: RELEASED FOR
EDIT'
##IN INBOX FOLDER
        pick -from $member -an -search 'Version: released for edit' -sequence picked >! hits

##PICK MESSAGE IF EDITED BY GEO
        pick -lb -cc $member -an -search 'Part-of-News-Letter:' -rb -sequence pickedup >! hit

##IF REMOVE EDIT FLAG IF UPDATED CONTRIBUTION RECIEVED FROM THIS MEMBER
        if ((! -z hit) || (! -z hits) && ($member != $user)) then

############
if (($member != "b.bala") && (! -z hits)) then
                awk '/Directory:/ {print $2 > "foladdr"}' findaddr$member
                set actualfold = `cat foladdr`
##IF THERE IS CONTRIBUTION FROM THIS MEMBER ON THIS CHECK THEN
##REMOVE EDIT CHECK AT MEMBERS DIRECTORY AND COMMON STORE
                rm Mail/newsloc/$member/checkedit $actualfold$odr/checkedit
endif
############
if (($member == "b.bala") && (! -z hits)) then
        rm Mail/newsloc/$member/checkedit
endif
endif
##IF CONTRIBUTION CARRIES "Version: released for edit" PARAMETER THEN
## PROCESS TO REMOVE THIS FROM HEADER
        if ! -z hits then
                awk '/Directory:/ {print $2 > "foladdr"}' findaddr$member
                set actualfold = `cat foladdr`

##REMOVE THE VERSION RELEASED FOR EDIT HEADER PARAMETER FROM THE HEADER
OF
##THE MESSAGE MAINTAINING SAME MESSAGE NUMBER
                scan picked -format "%(msg)" >! mnumbers
                set msgnumbers = `cat mnumbers`
                show $msgnumbers -form release.form >! newmsg
                rmm picked
                mv newmsg Mail/inbox/$msgnumbers
                refile $msgnumbers +newsloc
                rm mnumbers
        endif
end
```

```
#
pick +newsloc -sequence picked >! hits
if ! -z hits then
        refile all +inbox -src +newsloc >&! junk
endif
folder +inbox >&! junk
###************************************************************

echo linking contributions from a group member to all other group members
echo " "

while ( $i <= $#gr_lst )
#
##PICK ALL CONTRIBUTIONS FROM EACH GROUP MEMBERS WHICH ARE SENT
##AS PART OF NEWS LETTER
        pick -lb -from $gr_lst[$i] -an -search 'Part-of-News-Letter:' -rb -sequence picked +inbox >! hits
#
        pick -lb -cc $gr_lst[$i] -an -search 'Part-of-News-Letter:' -rb -sequence pickedup +inbox >! hit
        foreach member ( $gr_lst )
        if ! -z hit then
                if (($gr_lst[$i] != $member ) && ( $gr_lst[$i] != "$user") && ($member != "$user"))
then
##GET GROUP MEMBERS DIRECTORY PATH ON THEIR HOST
#######
if ($member != "b.bala") then
                awk '/Directory:/ {print $2 > "foladdr"}' findaddr$member
                set actualfold = `cat foladdr`
                refile pickedup -li +$actualfold$odr >&! junk
endif
########
                endif
        endif
##LINK ALL PICKED CONTRIBUTIONS TO ALL OTHER (THAN ORIGINATOR)
##GROUP MEMBERS AS READ ONLYAT THEIR HOST IN NEWSLOC FOLDER
        if ! -z hits then
                if (($gr_lst[$i] != $member ) && ( $gr_lst[$i] != "$user") && ($member != "$user"))
                then

##GET GROUP MEMBERS DIRECTORY PATH ON THEIR HOST
##############
if   ($member != "b.bala") then
                        awk '/Directory:/ {print $2 > "foladdr"}' findaddr$member
                        set actualfold = `cat foladdr`
                        refile picked -li +$actualfold$odr >&! junk

endif
                endif
        endif
        end #end of for loop
##FILE ALL CONTRIBUTION FROM THIS MEMBER IN HIS/HER FOLDER WITHIN COMMON
STORE
        if ((! -z hits) || (! -z hit)) then
                refile pickedup +newsloc/{$gr_lst[$i]} >&! junk
                refile picked +newsloc/{$gr_lst[$i]} >&! junk
##CHAGE ACCESS MORE TO READ ONLY TO ALL MEMBERS
                scan +newsloc/$gr_lst[$i] -format "%(msg)" >! chmodnum
                set chnum = `cat chmodnum`
                cd {$bdr}bala{$mdr}{$gr_lst[$i]}
                foreach changenum ($chnum)
                        chmod 655 $changenum
                end
```

```
                #echo " working with $gr_lst[$i]  "
                home
        endif
        @ i++
end #end of while loop
#
###

###*****************************************************************
echo "checking for editrequest from all group members"
echo " "

##PICK MESSAGE FROM THIS MEMBER WHICH HAS SUBJECT AS EDIT REQUEST
foreach member ( $gr_lst )

##SET FOLDER TO INBOX
        folder +inbox >&! /dev/null
        pick -from $member -an -subject 'editrequest' -an -search 'X-msgnum:' -sequence pickedup >! hits
#
        if ! -z hits then
##OTHERWISE PICK THE MESSAGE NUMBER REQUESTED
                show pickedup >! tmp
                awk '/X-msgnum:/ {print int($2) > "tm"}' tmp
                set snum = `cat tm` >&! junk
      foreach ssnum ($snum)

                if (-e Mail/newsloc/$member/checkedit) then
                        break
                else
##IF EDIT REQUEST MESSAGE FOUND THEN CHECK FOR EDIT CHECK FLAG
##IF IT EXISTS THEN REMOVE THIS REQUEST AND OUT
                        cp Mail/newsloc/$member/$ssnum Mail/newsloc/$member/9999
                        refile pickedup +newsloc/request
                        folder +newsloc/$member >&! junk
##CONSTRUCT THE DRAFT HAVING AN ADDITIONAL PARAMETER VERSION RELEASED
FOR EDIT
                        anno 9999 -component Version: -text 'released for edit' -nodate >&! junk

                        if ($member == "b.bala") then
                                echo -n "To: $member@email" >! Mail/draftrelease
                        else
                                echo -n "To: $member" >! Mail/draftrelease
                        endif
                        show 9999 -form head.form >>! Mail/draftrelease
                        show 9999 -form body.form >>! Mail/draftrelease
                        chmod 600 Mail/newsloc/$member/9999
                        rm Mail/newsloc/$member/9999
##SEND THE REQUIRED MESSAGE FOR UPDATION
                send draftrelease
##GET GROUP MEMBERS DIRECTORY PATH ON THEIR HOST
################################
if   ($member != "b.bala") then
                awk '/Directory:/ {print $2 > "foladdr"}' findaddr$member
                set actualfold = `cat foladdr`
##LOCK FOR FUTHER EDIT REQUEST TO THIS MEMBERS CONTRIBUTION
                echo $snum > $actualfold$odr/checkedit
                echo $snum > Mail/newsloc/$member/checkedit
else
                echo $snum > Mail/newsloc/$member/checkedit
endif
##############################
```

```
##REFILE REQUEST MESSAGE IN NEWSLOC/REQUEST FOLDER ????-->
                endif
                        rm tm >&! junk
                end
        endif
end ##end of for loop
###*****************************************************************

echo "checking for delivery failure"
echo " "

folder +inbox >&! /dev/null

##CHECK FOR MESSAGE AS DELIVERY FAILURE
pick -search 'Message_Type: Delivery Report' -an -search 'Part-of-News-Letter:' -sequence picked >! hits
if ! -z hits then
        show picked >! hit
scan picked

##IF THERE IS DELIVERY FAILURE MESSAGE THEN GET ITS INITIAL DELIVERY TIME
        awk '/Date:/ {print  $2 $3 $4 $5 $6 $7 > "intialtime"}' hit
        set ctime = `cat intialtime`
        folder +newsloc/out >&! junk

##GET MESSAGE NUMBER OF ALL OUT GOING MESSAGES
        scan -format "%(msg)" >! checkfail
        set msgtime = `cat checkfail`
        foreach ftime ($ctime)
        foreach eachmsg ($msgtime)
                show $eachmsg -noshowproc >! hit

##GET DELIVERY TIME OF EACH OF OUT GOING MESSAGE
                awk '/^D/{print $2 $3 $4 $5 $6 $7 > "findtime"}' hit
                set foundtime = `cat findtime`

##IF THE DELIVERY FAILURE MESSAGE TIME IS SAME AS THIS MESSAGE TIME
##THAN RESEND THIS MESSAGE
                        if ($foundtime == $ftime) then
                        show $eachmsg -form delivnote.form >! Mail/msgresend
                send msgresend
                endif
        end ##end of for loop
        end ##end of for loop
endif
#*****************************************************************
echo "checking for contributions overdue and sending reminders"
echo " "

##PROCESS TO GET LIST OF READER MEMBER FROM THE GROUP LIST
cp member.dat dtmp
awk '{print $1, $2}' dtmp > dtmp1
awk ' $2 == "reader"  {print $1 $2}' dtmp1 > dtmp2
set mstat = (`cat dtmp2`)
set sflag = " "


##ISSUE REMINDERS IF NECESSARY
foreach member ( $gr_lst )

##CHECK THIS MEMBER WITH READER MEMBER LIST
##IF FOUND SET TRUE FLAG
```

```
foreach memberstat ( $mstat )

if ($member != "b.bala") then
        if $memberstat == "$member@quipureader" then
          set sflag = t
          break
    endif
else
        if $memberstat == "$member@emailreader" then
          set sflag = t
          break
    endif
endif
end

##IF TRUE FLAG NOT SET THEN PROCESS FOR REMINDER
if $sflag != "t" then
##echo PROCESSING FOR reminder to $member

##IF REMINDER HEADER FILE NOT EXISTS THEN PROCESS FOR IT
        if ! -e Mreminder2 then
                echo creating Reminder file
                set tmpdata = ( `cat newsletter.dat` )
                echo "Subject: Reminder" > formrem1
                echo "Part-of-News-Letter: $tmpdata[2]" >> formrem1
                cat formrem1 formrem2 > Mreminder2
                rm formrem1
        endif

##CHECK EACH MEMBERS FOLDER FOR MESSAGE
##IF INDIVIDUAL FOLDER IN COMMON STORE IS EMPTY THEN
##PICK INITIAL MESSAGE TO THIS MEMBER FROM THE NEWSLOC/OUT FOLDER
        pick +newsloc/$member -sequence picked >! hits

        if -z hits then
##CHECK MESSAGE TIME WITH  REMINDER TIME
##IF MESSAGE TIME PASSES REMINDER TIME THEN SEND REMINDER
                scan picked -format "%(rclock{date})" >! remtime
                set remindertime = `cat remtime`
                foreach rtime ($remindertime)
                        if ($rtime > $checktime1) then
                        if ($member != "b.bala") then
                                echo "To: $member" >! Mreminder1
                        else
                                echo "To: $member@email" >! Mreminder1
                        endif
                                cat Mreminder1 Mreminder2 >! Mail/Mreminder
                                send Mreminder
                                rm Mreminder1
                        endif
                end
                folder +newsloc/{$member} >&! junk
                rm remtime
        endif

##PROCESS FOR SECOND REMINDER
if ! -z hits then
        pick +newsloc/reminder -to $member -sequence picked >! hit
        if ! -z hit then

##CHECK FIRST REMINDER TIME WITH  TO ISSUE SECOND REMINDER
```

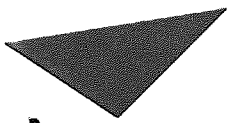
```
##IF MESSAGE TIME PASSES REMINDER TIME THEN SEND SECOND REMINDER
                    scan picked -format "%(rclock{date})" >! remtime
                    set remindertime = `cat remtime`
                    foreach rtime ($remindertime)
                            if ($rtime > $checktime2) then
#############
                    if ($member != "b.bala") then
                            echo "To: $member" >! Mreminder1
                    else
                            echo "To: $member@email" >! Mreminder1
                    endif
############
                                    cat Mreminder1 Mreminder2 >! Mail/Mreminder
                            #       send Mreminder
                                    rm Mreminder1
                            endif
                    end
                    folder +newsloc/{$member} >&! junk
                    rm remtime
            endif
    endif
endif
end ##end of for loop

##REMOVE ALL TEMPORARY REMINDER FILES
rm dtmp
rm dtmp1
rm dtmp2

###*****************************************************************
folder +inbox >&! /dev/null
##echo $gr_lst

##REMOVE TEMPORARYDIRECTORY ADDRESS FILES FOR ALL MEMBERS
foreach member ( $gr_lst )
if ($member != "b.bala") then
        rm findaddr$member
endif
end
unset gr_Lst i mdr odr
rm member_last
```

Appendix 6

Operation Manual

For Prototype

Editing a Newsletter by a Group



Aston University

Content has been removed for copyright reasons

Appendix 7

Testing

A 7.1 Testing Results

The process of testing and debugging a program is an implementation phase and involves verification and validation (Sommerville, 89). '*Verification involves checking that the program confims to its specification*'. '*Validation involves checking that the program as implemented meets the expectations of the user*'. Testing is the process of establishing the presence of faults. Debugging is concerned with finding and removing faults. Being a prototype, debugging may not be necessary at great length for this system.

There are three different type of testing techniques described (Sommerville, 89) which can be used in systematically testing a program. These are Equivalence partitioning, Structural (also called 'black-box') testing and testing of real-time systems. It is very difficult to say how far any particular testing technique is required in the case of testing a throw away prototype, but it does not need rigourous testing.

A 7.1.1 Prototype Testing

The testing must proceed in stages. There are five stages identified in the testing process (Sommerville, 89). These distinct stages are known as unit testing, module testing, sub-system testing, integration testing and acceptance testing. These phases are shown in figure 7.1. During the unit testing phase individual components i.e. functions and objects are tested to ensure that they operate correctly. The coding is tested in this phase.

In this project, each function and object is tested before incorporating it in a module. The test results for few functions (such as add member, send notification, and generate edit request)

are described later. Each unit is tested and modified until the desired functionality was achieved.

Module testing tests a group of functions and objects as a whole. For example a module 'create newsletter' which consists of various functions: define common store with individual folders, define member's record and define a Newsletter details, tests the functionality of the module as whole, including the side effects of individual functions. The test results of a few modules (create-newsletter, process contribution, accept contribution, history and statistics and generate Newsletter) are discussed later. The module testing is carried out until the desired functionality is attained.

Sub-system testing is carried out by testing each sub-system, which is made up of the various modules. The examples are testing of activity generator, activity monitor and activity responder as a whole. In this prototype, individual sub-system testing is not possible, because the functionality of the sub-systems are inter-linked. For example if a member sends an edit request working with activity responder, it is to be checked by the activity monitor to release the required version from the common store. Sub-system testing is taken care in the next phase (i.e integration testing).

Integration testing is carried out after the sub-systems are integrated to make up the entire system. This phase of testing involves the creation of six accounts, on two different hosts (both with X.400 implementation), for each of the six participants and installation of appropriate activity sub-systems on these accounts. The integration testing is concerned with finding errors which normally result from unanticipated interaction between sub-systems and components. In this case testing is concentrated on ensuring that the system provides the function specified in the requirements.

The acceptance testing (sometime called *stress testing* or *alpha testing*) exercises the system functions and demonstrates its performance. Conventionally the object of this testing is to allow the procurer to accept the system from a developer. The aim of acceptance testing in this research is to provide a system for communication with the users to obtain their feedback. In this regard an example of 'ASTON FORTNIGHT' Newsletter is considered to check various functions, viz edit request, new update, suggestion and accept contribution.

The system is tested as a whole, comprising activity generator, activity monitor and activity responder. The activity generator and activity monitor are installed at the account of the user who acts as group editing organiser. The activity responder is implemented on the rest of the accounts who act as group members. The testing is carried out by creating a Newsletter, sending notifications, submission of contributions by members of the group, sending editing request and releasing requested version from common store etc.

A 7.2 Testing Results

Add member:

Executable file ---> **add_member**
Record file for group member's ---> **member.dat**

Command line:
24 /isode/csam/bala->cat member.dat (contents of member.dat file **before** add operation)
tb1@quipu author networks x.400
bala@quipu geo news-round-up editing
bernard@quipu editor fundings computerisation
tb2@quipu author message-handling-system text-handling
25 /isode/csam/bala->add_member

The following questions are to obtain the details of a group member for this News Letter.

Enter e-mail address of this group member (username@hostname): **tb3@quipu**

status of group member: **author**

Subject area for this group member: **Aston-graduates**

Topic for this group member: **top-emloyment**

Is this information correct y/n ?? y

26 /isode/csam/bala->cat member.dat
tb1@quipu author networks x.400
bala@quipu geo news-round-up editing
bernard@quipu editor fundings computerisation
tb2@quipu author message-handling-system text-handling

tb3@quipu author Aston-graduates top-emloyment (**after** adding member tb3)

****

Send Notification:

Executable file ---> **notification**
Copy of notification is kept at ---> **newsloc/out**

Command line:
44 /isode/csam/bala->scan +newsloc/out (Contents of newsloc/out **before** notification)
1 11/12 To:tb1 networks<<This is a funding special issue. >>
2 11/12 To:bernard fundings<<This is to check the editing procedure

45 /isode/csam/bala->>notification

Enter Notification To (username@hostname): **tb3@quipu** (to be enter by GEO)

(complete header components are displayed by system)
To: tb3@quipu
Status: author
Subject: Aston-graduates
Topic: top-emloyment
Part-of-News-Letter: aston-fortnight-vol12-no24
Latest-Submission-Date: 15-12-92
Fcc: newsloc/out

The first notification allow you to start sending contribution. (first message to be typed)
The given date is very close.

What now? s (notification is sent to tb3 member carrying required information)

46 /isode/csam/bala->scan +newsloc/out (Contents of newsloc/out **after** notification)
1 11/12 To:tb1 networks<<This is a funding special issue. >>
2 11/12 To:bernard fundings<<This is to check the editing procedure
3+ 11/13 To:tb3 Aston-graduates<<The first notification allow yo
47 /isode/csam/bala->

****

Edit request:

Executable file ---> **xrequest**
Contribution folder ---> **newsloc/out**

(contents of newsloc/out folder)

1+ 06/26 Bala Buksh 021 35 X.500<<Directory services are implementaed on 'Q
2 06/26 To:bala@quipu X.500<<Directory services are implementaed on 'Q

Command line:
50 /isode/csam/bala-> xrequest

To: bala@quipu
Subject: editrequest
X-msgnum: **2**

Please release X-msgnumber for further updation.

--------Enter additional text

It is important to make some changes in the contribution. (optional text)

What now? send (request is sent for release version number 2 from common store)

51 /isode/csam/bala->

****

Create new activity (with a process to clsoe earlier):

Executable file ---> **defineshell**
Record file ---> **newsletter.dat**
Special Remark ---> **remark.dat**

Command line:
4 /isode/csam/bala->defineshell

Warning:-
---
1. This system supports only a single News Letter.

2. There is a News Letter currently being prepared, if you continue
the current News Letter will be closed.

3. If a News Letter is closed all old information will be available as follows:
(i) activity details --> oldnewsletter.dat
(ii) group member details --> oldmember.dat
(iii) remark details --> oldremark.dat
(iv) user names --> oldmember_name
(v) newsletter folder --> oldnewsloc

Do you want to create a New News Letter(and close old one) y or n ??

y --> To create a New News Letter.

q --> To Quit (without closing existing News Letter).

Enter Option y or q ---> ?? **y**

Are you sure y or q ---> ?? **y**

Following questions will collect the details for the activity.

`EDITING A NEWS LETTERS` (at present exists)

Name & Issue of the News Letter: **aston-fortnight-vol2-no5**

Date of Release: **1-1-93**

General Remarks if Any ? **This will be new year issue on admission policy.**

..?? Are all these details correct ? Y/N ?? **y**

5 /isode/csam/bala->cat newsletter.dat
Editing-News-Letters aston-fortnight-vol2-no5 1-1-93
6 /isode/csam/bala->cat remark.dat
This will be new year issue on admission policy.

(newsloc (common store) has no folder for members but control folders, just after creating activity)

```
7 /isode/csam/bala-> folders +newsloc   (Newly created common store)
          Folder     # of messages ( range ); cur  msg  (other files)
          newsloc+ has   no messages          ;          (others).
    newsloc/document  has   no messages.
       newsloc/out  has   no messages.
    newsloc/reminder  has   no messages.
     newsloc/request  has   no messages.

               TOTAL=    0 messages in 5 folders.
```

(Information of **old** newsloc moved to oldnewsloc)

```
8 /isode/csam/bala-> folders +oldnewsloc
          Folder     # of messages ( range ); cur  msg  (other files)
        oldnewsloc+ has   no messages          ;          (others).
     oldnewsloc/bala  has   no messages.
   oldnewsloc/bernard  has    1 message  (  1-   1); cur=   1.
  oldnewsloc/document  has    2 messages (  1-   2); cur=   1.
    oldnewsloc/oldtb2  has   no messages.
    oldnewsloc/oldtb3  has    1 message  (  1-   1).
      oldnewsloc/out  has    4 messages (  1-   4); cur=   4.
  oldnewsloc/reminder  has   no messages.
   oldnewsloc/request  has    2 messages (  1-   2).
      oldnewsloc/tb1  has    2 messages (  1-   2); cur=   2; (others).
      oldnewsloc/tb2  has    3 messages (  1-   3); cur=   3.
      oldnewsloc/tb3  has    1 message  (  1-   1).

               TOTAL=   16 messages in 12 folders.

9 /isode/csam/bala->

*****************************
```

Process Contribution:

Executable file ---> **process_contrib**

Command line:

```
33 /isode/csam/bala-> process_contrib  (displays all contributions from all members)

      contribution from tb1
  1  12/09 To:tb1        X.400<<This first notificatin. >>
  2  12/09 BalaTest1     X.400<<Message From geo :- this is first contrib
  3+ 12/14*Bala Buksh 021 35  X.400<<Message From geo :- this is first

      contribution from bernard
  1+ 12/11 To:bernard        AI<<This is first notification to you. >>

      contribution from balab
  1+ 12/11 To:balab@email        S/W<<This is first notification to you>>

      contribution from tb2
scan: no messages in newsloc/tb2
```

contribution from tb3
1 12/15 To:tb3 osi-networks<<This is the first notification. the>>

---> a to accept contribution

---> e to edit contribution

---> q to quit

Enter ----------> ???? **a (accept contribution)**

Carefully type in the name of the originator of the contribution ?? **tb1**

tb1 s contribution (contributions from tb1 are displayed again)

1 12/09 To:tb1 X.400<<This first notificatin. >>

2 12/09 BalaTest1 X.400<<Message From geo :- this is first contrib

3+ 12/14*Bala Buksh 021 35 X.400<<Message From geo :- this is first

newsloc/tb1+ has 3 messages (1- 3); cur= 3.

Select contribution number ?? **3** (contribution **3** is accepted)

(accepted contribution displayed and a copy send to the author of the contribution)
To: Buksh.Bala@quipu.aston.ac.uk
From: Bala Buksh 021 359 3611 X4272 <Buksh.Bala@quipu.aston.ac.uk>
Status: author
Subject: X.400
Topic: mhs
Version: Accepted
Part-of-News-Letter: Aston-fortnight-vol30-no6
Latest-Submission-Date: 7-1-93
Updated-On: Mon Dec 14 22:47:13 GMT 1992

---

his is first contribution

This first notificatin.

adding to the first not.

34 /isode/csam/bala->

****

History and Statistics:

Executable file ---> **his_stat_new**

Command line:
14 /isode/csam/bala->his_stat_new

Application: Editing-News-Letters
News Letter: Aston-fortnight-vol30-no6
Completion Date: 15-1-93
Group Editing Organiser: bala@quipu

Members Details:-
Authors:
tb1@quipu
tb3@quipu

Sub-Editors/Coordinators:
bernard@quipu
tb2@quipu

Readers:

Overall Contribution Statistics :-
Total No. of Members in the Group = 5

Member's name	Notification /sent yes/no	Total Contributions	Number of Suggestions	Accepted yes/no
tb1@quipu	yes	3	-	yes
tb3@quipu	yes	1	-	no
bernard@quipu	yes	1	-	no
tb2@quipu	no	-	-	no

15 /isode/csam/bala->

****

Generate document:
The shecll script provided within the main module under option 'g'.
The option needs to enter the sequence number of the accepted contributions.

(All accepted versions are displayed before asking to enter sequence numbers.)

1+ 12/09 BalaTest1 X.400<<Message From geo this is first contrib

2 12/15*Bala Buksh 021 35 X.400<<Message From geo this is first

Please enter the numbers (separated by spaces) of the contributions in the order they will appear in the New Letter ?? **2 1**

(Displaying format for the Newsletter)
News Letter : Aston-fortnight-vol30-no6 Release Date: 15-1-93
Main Editor: bala@uk.ac.aston.quipu

Subject: X.400
Topic: mhs
From: Bala Buksh 021 359 3611 X4272 <Buksh.Bala@quipu.aston.ac.uk>
---
this is first contribution
This first notificatin.
adding to the first not.

Subject: X.400
Topic: mhs
From: BalaTest1 <tb1@quipu.aston.ac.uk>
---
this is first contribution
This first notificatin.
adding to the first not.

(After generating the document it exits from the prototype system.)
30 /isode/csam/bala->

****

Appendix 8

Abstract Syntax Notation One

A 8.1 Abstract Syntax Notation One

OSI's method of specifying abstract objects is called Abstract Syntax Notation One (ASN.1). ASN.1 is flexible notation that allows one to define a variety data types. The following meta-syntax is used in describing ASN.1 notation (Marshall, 89):

BIT monospace denotes literal characters in the type and value notation

n1 bold italics denotes a variable

[ ] bold square brackets indicate that a term is optional

{ } bold braces groups related terms

| bold italics bar delimits alternatives with a group

... bold ellipsis indicates repeated occurrences

= bold equal sign expresses terms as sub-terms

-- pair of hyphens to comment.

Identifiers (names of value and fields) and type references (names of types) consists of upper and lower case letters digits, hyphens and spaces. Identifiers begin with lower case letters, type references begin with upper case letters.

In ASN.1, a type is a set of values. ASN.1 has four kinds of type : simple types, which are 'atomic' and have no components; structured types, which have components; tagged types, which are derived from other types; and other types, which include CHOICE and ANY types (Marshall, 89). Types and values can be given names with the ASN.1 assignment operator (::=).

There are four classes of tag:

i) universal --> 0 (whose meaning is same in all applications)

ii) application --> 1 (whose meaning is specific to an application)

iii) context-specific --> 2 (whose meaning is specific to a given structured type)

iv) private --> 3 (whose meaning is specific to an enterprise).

Table A 8.1 lists some ASN.1 types and their universal-class tags.

Type	Tag Number
INTEGER	2
BIT STRING	3
OCTET STRING	4
NULL	5
OBJECT IDENTIFIER	6
SEQUENCE and SEQUENCE OF	16
SET and SET OF	17
Printable String	19
IA5String	22
UTTime	23

Table A 8.1

A 8.1.1 SAGE Environment

This module specifies the fundamental components within the SAGE project environment, which are users, services and the ports through which they communicate (Benford et.al., 90).

```
SageEnvironmentRefinement {
        ccitt(0)asdc(2)aston(???)modules(0)sage(0) sage-env-refinement(1) }

Sage DEFINITIONS::=
        BEGIN
                --Prologue
```

```
EXPORTS
        sage-environment, sage-environment-refinement, sage-system, sage-user;
IMPORTS
--Sage Abstract Service
        sage-admin, sage-retrieve, sage-store

        FROM SageAbstractService {
                ccitt(0)asdc(2)aston(???)modules(0)sage(0) sage-abstract-service(2) }

--Sage Object Identifiers
        id-ot-sage-environment, id-ot-sage-system, id-ot-sage-user,
        id-ref-sage-environment

        FROM SageObjectIdentifiers {
                ccitt(0)asdc(2)aston(???)modules(0)sage(0) objectidentifier(3) }

        Certificates
                FROM AuthenticationFramework {
                        joint-iso-ccitt ds(5) modules(1) authentication-framework(7) }

--Abstract Service Notation
        OBJECT DEFINE
        FROM AbstractServiceNotaion {
                joint-iso-ccittmhs-motis(6)asdc(2)modules(0)notation(1) }

--Sage Environment
        sage-environment OBJECT
        ::= id-ot-sage-environment

--Sage Environment Refinement
        sage-environment-refinement REFINE sage-environment AS
        sage-user RECURRING
```

```
        sage-system
                sage-admin[S]          PAIRED WITH sage-user
                sage-retrieve[S]       PAIRED WITH sage-user
                sage-store[S]          PAIRED WITH sage-user
        ::= id-ot-sage-environment

--PORTS
        sage-user OBJECT
        PORTS {
                sage-admin[C],
                sage-retrieve[C],
                sage-store[C]  }
        ::= id-ot-sage-user

        sage-system OBJECT
        PORTS {
                sage-admin[S],
                sage-retrieve[S],
                sage-store[S]
        ::= id-ot-sage-system

--port types
        sage-admin PORT
                CONSUMER INVOKES {
                RegisterMember,
                De-RegisterMember,
                RegisterActivity  }
        ::= id-pt-sage-admin

        sage-retrieve PORT
                CONSUMER INVOKES {
                EditRequest,
                ListGroup,
                ListStatistics,
                EditSuggestion,
                ShowDifference,
                ReadContribution,
```

```
            EditReminder  }
        ::= id-pt-sage-retrieve

        sage-store PORT
            CONSUMER INVOKES {
            CreateActivity,
            EditNotification,
            UpdateDocument,
            RenameMember,
            DeleteContribution
            AcceptContribution,
            GenerateDocument  }
        ::= id-pt-sage-store

--Operations
--Abstract Bind Operation
sage-bind ::=  ABSTRACT-BIND
            TO {sage-manage[S]}
            BIND
                ARGUMENT        Bind-arguments
                RESULT          Bind-results
                BIND-ERROR      SET {
            [1] ENUMERATED {
                busy, authentication-error, unacceptable-security-context },
            [2] Errors }
        ::= 0

--Abstract Unbind Operation
sage-unbind ::=  ABSTRACT-UNBIND
              FROM {sage-manage[S]}
        ::= 1

RegisterMember ABSTRACT-OPERATION
        ARGUMENT        RegisterMember-Argument
        RESULT          NULL
        ERRORS          {accessControlError, objectError}
        ::= 2
```

De-RegisterMember ABSTRACT-OPERATION

ARGUMENT	DistinguishedName
RESULT	De-RegisterMember-Result
ERRORS	{accessControlError, objectError}
::= 3	

RegisterActivity ABSTRACT-OPERATION

ARGUMENT	Create-Argument
RESULT	NULL
ERRORS	{accessControlError,objectError,schemaViolationError, attributeError}
::= 4	

EditRequest ABSTRACT-OPERATION

ARGUMENT	Request-Arguments
RESULT	NULL
ERRORS	{accessControlError, objectError}
::= 5	

ListGroup ABSTRACT-OPERATION

ARGUMENT	DistinguishedName
RESULT	Group-Result
ERRORS	{accessControlError, objectError}
::= 6	

ListStatistics ABSTRACT-OPERATION

ARGUMENT	DistinguishedName
RESULT	Statistics-Result
ERRORS	{accessControlError, objectError,attributeError}
::= 7	

EditSuggestion ABSTRACT-OPERATION

ARGUMENT	ObjectName
RESULT	NULL
ERRORS	{accessControlError, objectError}
::= 8	

```
ShowDifference ABSTRACT-OPERATION
        ARGUMENT        Difference-Argument
        RESULT          Difference-Result
        ERRORS          {accessControlError, objectError}
        ::= 9

ReadContribution ABSTRACT-OPERATION
        ARGUMENT        Read-Argument
        RESULT          Read-Result
        ERRORS          {accessControlError, objectError}
        ::= 10

EditReminder ABSTRACT-OPERATION
        ARGUMENT        DistinguishedName
        RESULT          NULL
        ERRORS          {accessControlError, objectError}
        ::= 11

EditNotification ABSTRACT-OPERATION
        ARGUMENT        EditNotification-Argument
        RESULT          NULL
        ERRORS          {accessControlError, objectError,schemaViolationError,
                        attributeError}
        ::= 12

UpdateDocument ABSTRACT-OPERATION
        ARGUMENT        Update-Argument
        RESULT          UpdateDocument-Result
        ERRORS          {accessControlError, objectError,schemaViolationError,
                        attributeError}
        ::= 13
```

```
RenameMember ABSTRACT-OPERATION
        ARGUMENT        Rename-Argument
        RESULT          NULL
        ERRORS          {accessControlError, objectError,schemaViolationError,
                        attributeError}
        ::= 14

DeleteContribution ABSTRACT-OPERATION
        ARGUMENT        DeleteContribution-Argument
        RESULT          NULL
        ERRORS          {accessControlError, objectError,attributeError}
        ::= 15

AcceptContribution ABSTRACT-OPERATION
        ARGUMENT        ObjectName
        RESULT          NULL
        ERRORS          {accessControlError, objectError}
        ::= 16

GenerateDocument ABSTRACT-OPERATION
        ARGUMENT        Generate-Argument
        RESULT          Generate-Result
        ERRORS          {accessControlError, objectError}
        ::= 17

--Abstract Errors
accessControlError ABSTRACT-ERROR
        PARAMETER       AccessControlError
        ::=0

objectError ABSTRACT-ERROR
        PARAMETER       ObjectError
        ::=1

attributeError ABSTRACT-ERROR
        PARAMETER       AttributeError
        ::=2
```

```
schemaViolationError ABSTRACT-ERROR
        PARAMETER       SchemaViolationError
        ::=3

--arguments
Bind-arguments ::= CHOICE {
        NULL,                   --if no authorisation is required
        [1] SET {               -- if authorisation is required
                user-name               [0]     SageName,
                initiator-credentials   [1]     Initiator-Credentials,
                security-context        [2]     SecurityContext     OPTIONAL } }

RegisterMember-arguments ::= SET {
        userName                [0]     DistinguishedName,
        activityName            [1]     ObjectName,
        attributes              [2]     SET OF AttributeType }

Create-arguments ::= SET {
        name            [0]     DistinguishedName,
        domainName      [1]     ObjectName,
        attributes      [2]     Attributes }

Request-arguments ::= SET {
        name            [0]     DistinguishedName,
        requestName     [1]     ObjectName }

Show-Difference-arguments ::= SET {
        name            [0]     ObjectName,
        attributes      [1]     SEQUENCE OF ObjectType }

Read-arguments ::= SET {
        name            [0]     ObjectName,
        attributes      [1]     SET OF AttributeType }

EditNotification-arguments ::= SET {
        name            [0]     DistinguishedName,
        attributes      [1]     SET OF AttributeType }
```

```
Update-Document-arguments ::= SET {
        name            [0]     DistinguishedName,
        attributes      [1]     SEQUENCE OF Modification }

GenerateDocument-arguments ::= SET {
        objectList      [0]     SEQUENCE OF ObjectName,
        errors          [1]     accessControlError, objectError }

DeleteContribution-arguments ::= SET {
        name            [0]     ObjectName,
        errors          [1]     accessControlError, objectError, attributeError }

Rename-arguments ::= SET {
        oldname         [0]     DistinguishedName,
        newname         [1]     DistinguishedName,
        attributes      [2]     SET OF AttributeType,
        redirect        [3]     BOOLEAN    DEFAULT    TRUE  }

--results
Bind-result ::= CHOICE {
        NULL,                   --if no authorisation is required
        [1] SET {               -- if authorisation is required
                user-name               [0]     SageName,
                responder-credentials   [1]     ResponderCredentials } }

De-RegisterMember-result ::= SEQUENCE {
        name                    [0]     DistinguishedName,
        targetAttributes        [1]     SET OF AttributeType,
        errors                  [2]     Errors }

List-group-result ::= SEQUENCE {
        clusterName             [0]     DistinguishedName,
        clusterContents         [1]     SEQUENCE OF DistinguishedName,
        attributes              [2]     Attributes }
```

```
List-statistics-result ::= SEQUENCE {
        name                [0]     DistinguishedName,
        contents            [1]     SEQUENCE OF ObjectName,
        attributes          [2]     Attributes }

Show-difference-result ::= SEQUENCE {
        name                [0]     DistinguishedName,
        targetAttributes    [1]     Attributes,
        errors              [2]     Errors }

Read-result ::= SET {
        name                [0]     DistinguishedName,
        targetAttributes    [1]     Attributes,
        errors              [2]     Errors }

UpdateDocument-result ::= SET {
        name                [0]     DistinguishedName,
        Attribute           [1]     UTTime,

GenerateDocument-result ::= SEQUENCE {
        name                [0]     SEQUENCE OF ObjectName,
        values              [1]     SEQUENCE OF AttributeValue,
        errors              [2]     Errors }

--errors
AccessControlError    ::= {readOnly, readWriteOnly, writeOnly, notAccessible,
                           permissionDenied }
AttributeError        ::= ANY
ObjectError           ::= ANY
SchemaViolationError ::= ANY

--definitions
Attributes ::= SET OF Attribute
Attribute ::= SEQUENCE {
        type    [0]     AttributeType,
        values  [1]     SET OF values }
```

```
AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY

Errors ::= SET OF Error
Error ::= ANY

Modification ::= CHOICE {
        add     [0]     attribute,
        delete  [1]     SEQUENCE OF delete-arguments }

Delete-arguments ::= SEQUENCE {
        type    [0]     AttributeType,
        value   [1]     SET OF AttributeValue }

DistinguishedName ::= RDNSequence

SageName ::= IA5String

ObjectName ::= CHOICE OF {RDNSequence}

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET OF AttributeValueAssertion

AttributeValueAssertion ::= SEQUENCE OF { AttributeType, AttributeValue }

ObjectType ::= OBJECTIDENTIFIER

InitiatorCredentials ::= CHOICE {
        simple  [0]     Password,
        strong  [1]     StrongCredentials (WITH COMPONENTS {
                        ...,
                        binb-token PRESENT })}
```

```
ResponderCredentials ::= CHOICE {
        simple  [0]     Password,
        strong  [1]     StrongCredentials (WITH COMPONENTS {
                        binb-token PRESENT })}

Password ::= CHOICE {
        IA5String (SIZE (0..ub-password-length)),
        OCTET STRING (SIZE (0..ub-password-length)) }

StrongCredentials ::= SET {
        bind-token      [0]     Token           OPTIONAL,
        certificates    [1]     Certificates    OPTIONAL }

Token ::= ANY

SecurityContext ::= SET OF SecurityLabls (SIZE (1..ub-security-labls))

SecurityLabls ::= ANY

END --SageAbstractService
```