Journal of the Industrial Technology Research Institute.
Korea Maritime University. Vol. 14. 1997.

A Group Communication Platform for Cooperative Works

Jae-Hong Yim

공동 작업을 위한 그룹 통신 플랫폼

임 재 홍

Abstract

A GCP(Group Communication Platform) is proposed as a group communication protocol for supporting cooperative works among multiple users geographically dispersed in networks. Some of the most important requirements for groupware system are to maintain consistency among shared objects processed by group-work participants and to be implemented as an open system. Considering these, the GCP is designed to provide multicast service and lock mechanism for maintaining consistency among shared objects, and it is implemented on the OSI(Open Systems Interconnection) application layer by extending existing point-to-point ASEs(Application Service Elements) and developing new ASE named MSE(Multicast Service Element). In practice, the GCP is tested as group communication protocol for a collaborative document manipulation prototype system.

This paper describes overall configuration, services and communication primitives of the GCP for supporting cooperative works.

Keywords : group communication, GCP(Group Communication Platform), multicast, CSCW(Computer Supported Cooperative Work)

1. Introduction

As PCs and workstations spread since 1980's, they have been used for processing personal tasks more efficiently and conveniently. As their typical type of applications was for single user, however, the applications,

* Department of Electronics and Communication Engineering, Korea Maritime University.

such as wordprocessor, spread sheet, drawing tool etc., were developed for supporting personal tasks, and it was not considered that tasks in organizations was processed by coordination of organization's members[1].

The progress of computing powers and data communication techniques exert the important role on increasing interests in computer support of the cooperative work. This trend advanced toward the new field called groupware or CSCW(Computer Supported Cooperative Work) which is aimed to support more efficient and convenient group-work among users with a common goal. Furthermore, diverse CSCW applications, such as conferencing system, collaborative editing system, decision-supporting system, have been proliferated rapidly. As a part of research related to CSCW, this paper presents a design and implementation of a GCP(Group Communication Platform) for supporting cooperative works.

With the ultimate goal of providing support for cooperative works over communication network, the following design objectives were established for the GCP.

- The GCP should be independent of application and the type of data transported.
- It should provide services for maintaining consistency among shared objects processed by group-work participants.
- It should support the one-to-many, many-to-one, many-to-many forms of multicast transmission.
- It should be desinged as an open system in order to achieve interoperability with various applications, therefore it should be implemented on the basis of international(ISO/ITU-T) standards as soon as possible.
- It should be so designed as to make use of existing single-user applications with a little modification as a CSCW application.

Considering the above objectives, the GCP is designed to provide multicast service and lock mechanism for maintaining consistency among shared objects, and it is implemented on the OSI(Open Systems Interconnection) application layer by extending existing point-to-point ASEs(Application Service Elements) and developing new ASE named MSE(Multicast Service Element). In practice, the GCP is tested as group communication protocol for a collaborative document manipulation prototype system. As a whole, this paper describes overall configuration, functions and service primitives of the GCP.

The major advantages of the GCP are as follows.

- The modification of operating system kernel is not needed because the GCP is implemented on the OSI application layer.
- Existing single-user applications can be used on the GCP as a CSCW application with a little modification.

The remaining parts of this paper are organized as follows. Section 2 briefly describes a comparison of the GCP with a selected number of related works. Section 3 describes existing ASEs which form the basis of the GCP. Section 4 contains a description of a design of the GCP. Section 5 describes an implementation

of the GCP, the GCP applications. The last section 6 concludes this paper.

2. Related Work

Group or multicast communication has received considerable attention[2-6, 15] in distributed application design.

There are two well-known approaches to implement multicast service. One is a centralized approach, in which a central server multicasts data to all participants, whereas the other is a distributed approach that all the participants make connections with each other and each of them multicasts data to the others. Concerning implementation approach for multicast, the centralized approach is easier than the distributed approach, so most implementations are made through the centralized approach[2,4]. In this paper, however, the distributed approach is selected for flexibility although the increase of the number of connections, since the centralized approach is ineffective if the machine where the multicast server is executing fails.

In addition to two approaches to implement multicast service, multicast layer to be imlemented is another issue because it is possible to implement multicast facility in application, transport or network layer. Since implementation of multicast in transport[5] or network layer requires system-level modification, this is used for high-speed communication such as multimedia conferencing. Therefore, in most case multicast is implemented in application layer[2-4], furthermore ITU-T proceeds to develop international standard for multicast in application and recommends T. MCS(Multipoint Communication Service)[15].

Like the GCP, multicast remote procedure call[3] is implemented in application layer through the distributed approach, but this RPC makes use of unicast repeatedly, whereas the GCP uses MSE service in order to increase the degree of concurrent processing.

Our GCP does not guarantee ordered delivery in contrast to reliable group communication mechanism[6], however ordered delivery in the GCP may be used if there is a central server.

3. Background

In an open systems environment, the application layer consists of multiple protocol entities, each of which known as an ASE. Since a number of functions are common to many applications, the approach adopted by ISO(International Standards Organization) is to implement such functions as seperate ASEs and then to link them with selected application-specific ASEs when appropriate support service is required. The combined entity, known as an AE(Application Entity), is linked with the user application process[7]. Figure 1 shows the typical examples of ASEs.

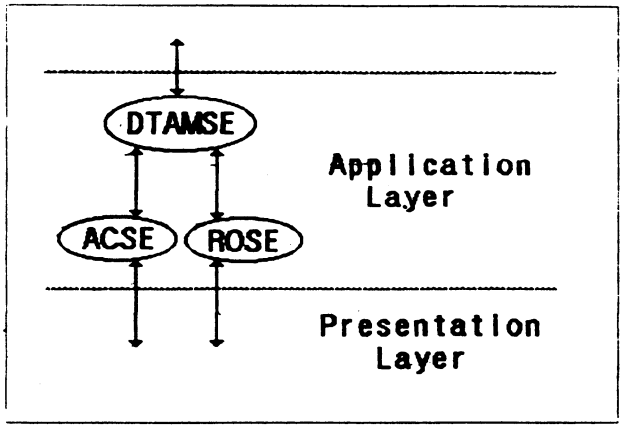


Figure 1. Examples of ASEs

The DTAMSE(Document Transfer And Manipulation Service Element) provides a common communication platform useful in telematic services as well as many other applications including multimedia services[8-10]. The DTAMSE has communication features such as document bulk transfer and document manipulation. DTAM document bulk transfer is used for conveying documents which are ODA(Open Document Architecture) and/or other types of documents to the remote system, and the document manipulation is used for remote document manipulation such as deleting, modifying and moving structured parts of the document.

A logical connection between two AEs is known as an association. The ASE that initiates the setting up and clearing of an association between two application specific ASEs is thus know as the ACSE(Association Control Service Element)[13]. After the association is set up, the AE may request an operation which will be performed by a remote peer AE. The AE receiving the operation responds the operation-result. To facilitate this type of the operation, the ROSE(Remote Operation Service Element) is defined. It provides a general framework of request/response facilities over which any application operation can be invoked by one ASE so as to be performed by the peer[14].

If the existing ASEs described above are used with no modifications as a group communication protocol for supporting cooperative work, the following problem will happen. The OSI model provides a point-to-point communication function, so the interface between two adjacent layers is established in a point-to-point manner. Consequently, in order to exchange data among group-work participants, as many specific application service entities as participants must be created, and then point-to-point connections between these and lower service entities must be established as depicted in Figure 2.

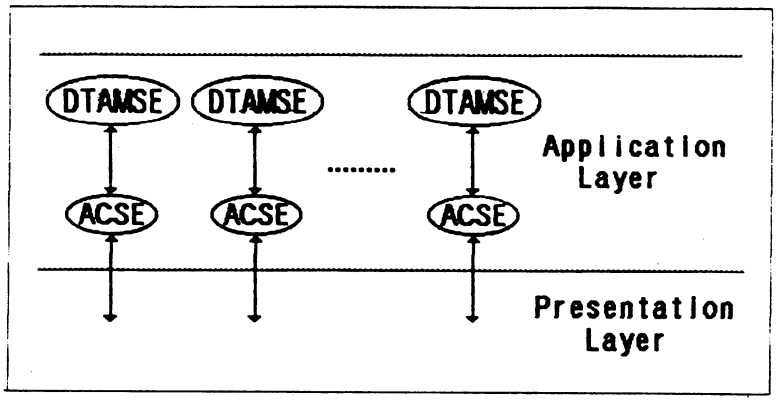


Figure 2. Problem of point-to-point interface

4. Design of the GCP

4.1 Design issues and requirements

The GCP for supporting the cooperative work should satisfy the following requirements.

(1) Multicast service

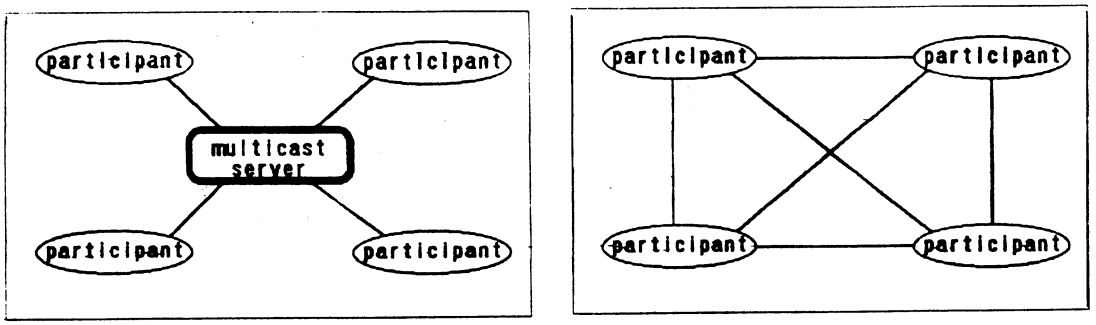
For maintaining consistency among shared objects processed by group-work participants, the status of shared objects modified by a participant should be transferred to all participants. Therefore, the GCP should provide various multicast transmission services such as one-to-many, many-to-one and many-to-many way. There are two well-known approaches to implement multicast service. One is a centralized approach, in which a central server multicasts data to all participants as shown in Figure 3(a), whereas the other is a distributed approach that all the participants make connections with each other and each of them multicasts data to the others as shown in Figure 3(b).

In this paper, the distributed approach is selected for flexibility although the increase of the number of connections, since the centralized approach is ineffective if the machine where the multicast server is executing fails.

(2) Lock Mechanism

Lock mechanism provides an exclusive access to shared objects. There are also two ways to implement lock mechanism, a centralized and a distributed method. The former may not provide a valid locking operation

by an unexpected fault of a server, so the latter is selected.



(a) Centralized approach (b) Distributed approach

Figure 3. Two approaches for multicast

(3) Use of existing single-user applications

In order to use existing single-user applications with a little modification as a groupware system, the GCP is designed to provide the applications with an API(Application Program Interface).

4.2 Configuration model

A configuration model for the GCP is depicted in Figure 4, in which the EDTAMSE(Extended DTAMSE) is an extension of DTAMSE for use as a collaborative document manipulation system[11, 12] and MSE(Multicast Service Element) is designed to provide services such as multiple associations managemnt(establishment and release) and multicast transmission.

MSE is a new ASE supplemented in this paper in order to work out the problem of point-to-point connections. The service primitives of ACSE and ROSE are used as it is, however their parameters are so modified a little as to handle group-identifier for communication among group-work participants.

The GCP is located at the OSI application layer, as illustrated in Figure 4.

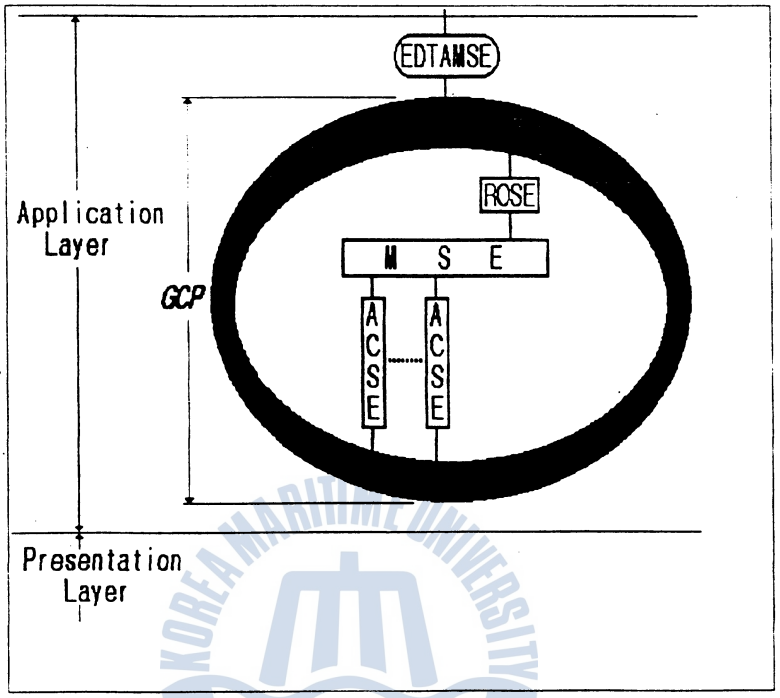


Figure 4. Configuration model for the GCP

4.3 Service primitives and parameters

Table 1 shows service primitives and parameters of EDTAMSE.

Table 1. EDTAMSE service primitives

Service Type	Primitives and Parameters
Association Control	**D_InitiateReq(dinfo, minfo, appl_ctx, calling_ae, called_ae)** **D_InitiateResp(dinfo, minfo, appl_ctx, responding_ae, result)** DEInfo *dinfo; /* structure which is needed for association between two EDTAM AEs */ MEInfo *minfo; /* structure which is composed of group_id and information about participants */ char *appl_ctx; /* application context */ AEInfo *calling_ae, *called_ae, *responding_ae; int result;

Service Type	Primitives and Parameters
Association Control	**D_TerminateReq(group_id, user_data)** **D_TerminateResp(group_id, user_data)** char *group_id, /* group identifier */ *user_data : /* user data from higher layer */
Document Manipulation	**D_DocumentOpenReq(group_id, document_id)** **D_DocumentCloseReq(group_id, document_id)** **D_LockReq(group_id, object)** **D_LockResp(group_id, object_id, result)** **D_UnlockReq(group_id, object)** **D_CreateReq(group_id, object, destination, position, content)** **D_DeleteReq(group_id, object)** **D_ModifyReq(group_id, object, selection, modification, content)** int document_id ; /* document identifier */ M_Object *object ;/* Manipulation Object structure */ *destination ; /* parent object of object to be created */ int position ; /* entry position among children of destination object */ char *content ; /* object content */ int selection ; /* modification selection of attribute or content */ struct Modif *modification ; /* attribute type and value to be modified */

〈Note〉 : DEInfo, MEInfo, AEInfo, M_Object structures are described in detail at appendix

Existing DTAMSE, which provides a point-to-point communication, is so extended as to support many-to-many forms of multicast transmission. For this, group-identifier for group communication is added to the parameters of DTAMSE, and MSE service is used instead of ACSE for association establishment among EDTAM AEs that share document. In this paper, DTAMSE is chosen for a collaborative document manipulation prototype system because it will soon provide a powerful communication platform as ITU-T standard protocol useful in multimedia services such as AVIS(AudioVisual Interactive Service) and AGCS(AudioGraphics Conferencing Service)[11, 12].

In Table 1, M_Object structure is defined as the structured parts of document(e.g., paragraph, figure) to be manipulated remotely, and can be represented as hierarchical tree structure. According to select_mode of M_Object structure(see appendix), manipulation object is either limited to one node indicated by MO or can include all nodes in the subtree of the selected node indicated by MO', as shown in Figure 5.

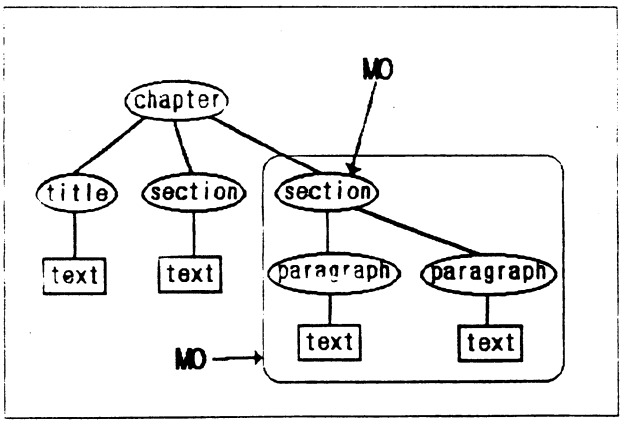


Figure 5. Manipulation object

Table 2 shows service primitives and parameters of MSE. When establishing or releasing association, multiple associations control primitives are interfaced and operated with EDTAMSE and ACSE, above and below of MSE respectively. Multicast primitive related to remote manipulation is designed to operate with ROSE.

Table 2. MSE service primitives

Service Type	Primitives and Parameters
Multiple Associations Control	**M_AssociateReq(minfo, appl_ctx, calling_ae, called_ae, user_data)** DINQ_apdu *user_data; /* PDU of D_InitiateReq */ **M_AssociateResp(minfo, appl_ctx, responding_ae, user_data)** DINQ_apdu *user_data; /* PDU of D_InitiateResp */
	M_ReleaseReq(group_id, reason, user_data) DTEQ_apdu *user_data; /* PDU of D_TerminateReq */ int reason; /* reason of association release */ **M_ReleaseResp(group_id, reason, user_data)** DTER_apdu *user_data; /* PDU of D_TerminateResp */
Data Multicasting	**M_SendData(group_id, class, member_name, user_data)** int class; /* Multicast of Unicast */ char *member_name; /* used only in case of unicast */ struct M_SenD *user_data; /* user data from ROSE */

Table 3 illustrates a mapping relationship among ASEs. As shown in this table, the association control of

EDTAMSE and MSE uses the services of ACSE, and the document manipulation of EDTAMSE uses the services of ROSE and M_SendData primitive of MSE.

Table 3. A relationship among ASEs

EDTAMSE	ROSE	MSE	ACSE
D_Initiate D_Terminate		M_Associate M_Release	A_Associate A_Release
D_Lock D_Unlock D_Create D_Delete D_Modify	RO_Invoke RO_Result RO_Error RO_U_Reject	M_SendData	

Figure 6 shows state transition diagram for the service elements of the GCP. The possible states of the service elements are shown in circles, and arrows indicate the possible transitions between the states using request, indication, response and confirm primitives. Upper side of the figure illustrates state transitions in the case of group-work initiator. On the other hand, lower side shows state transitions in the case of group-work responders.

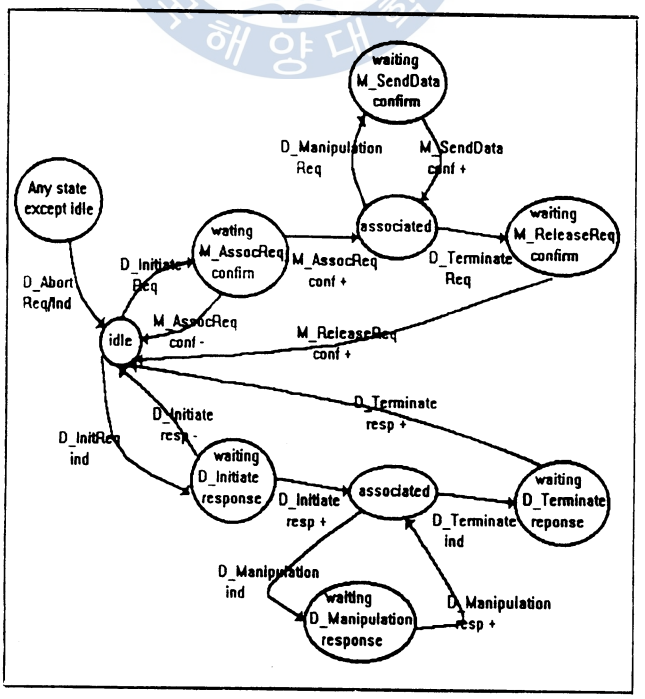


Figure 6. State transition diagram

5. Implementation and Applications of the GCP

5.1 Environment and system configuration

A collaborative document editing environment is composed of three SUN SPARCstations connected through 10Mbps Ethernet, as shown in Figure 7. In this figure, the GCP locates directly above TCP/IP without presentation and session layer to minimize communication overhead, and EDTAM service is used for a collaborative document manipulation prototype system. EDTAM daemon process is a server process in listening mode waiting a request for association establishment from any participant.

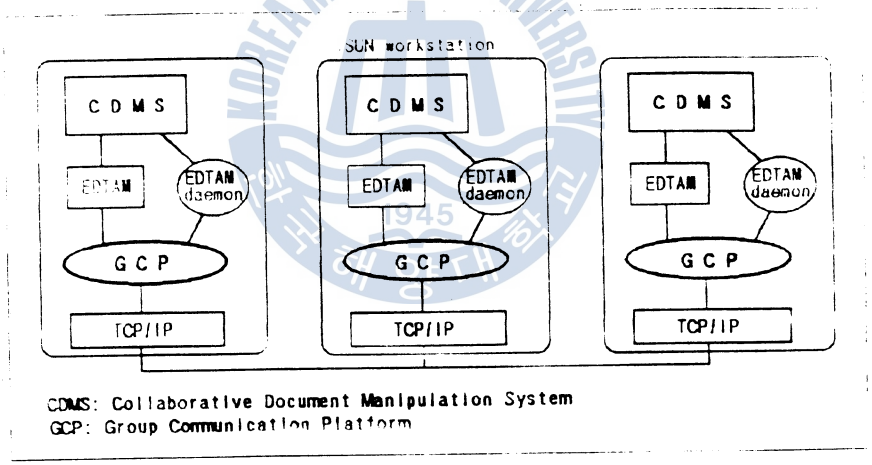


Figure 7. System configuration

Here, the following assumptions about group membership are made to simplify implementation of the GCP.
i) The information for group-work participants(name, address, port number, etc.) is stored in group-table, and this table is managed by CSCW application(e.g., CDMS in the case of Figure 7).
ii) Organization of group membership is categorized into two types. One is a static group where the group membership never changes, the other is a dynamic group where members may join or leave at any time. Only is the static group assumed in this paper, however the modification of group-table according to the changes of group member in the case of dynamic group is possible by using M_SendData primitive of MSE.

5.2 MSE services

MSE services are largely divided into multiple associations management and multicast transmission, and their algorithms are illustrated in this section.

(1) Multiple associations

Figure 8 shows an algorithm for multiple associations establishment. In this algorithm, MSE first receives the information for group-work participants from EDTAMSE, then forks as many child processes as participants and tries to establish multiple associations using ACSEs. In order to increase the degree of concurrent processing of multiple associations, the fork system call is invoked.

```
Algorithm for M_AssociateRequest primitive
input : group_id, group_table from MEInfo of parameters
output : return value(OK or NOTOK)
{
  initialize entry_number from group_table ;
  fork as many child processes as entry_number(i.e., number of group members) ;

if (parent process)
    for( ; ; )   /* wait for responses from ALL child processes */
    {
    if (receive OK responses form ALL child processes)
      return OK ;
    else if (receive responses form ALL child processes, but there is NOTOK)
      return NOTOK ;
 else
      continue ;
  }
else  /* child process */
  {
  make MARQ_PDU(PDU of M_AssociateRequest) ;
  call A_AssociateRequest primitive with MARQ_PDU ;
  if (return value of A_AssociateRequest is OK)
    set the association state of this member to OK ;  /* response to parent */
  else
    set the association state of this member to NOTOK ;  /* response */
  exit(0) ;
  }
}
```

Figure 8. Algorithm for multiple associations

(2) Multicast transmission

After multiple associations establishment, messages can be multicasted to group-work paricipants with M_SendData primitive. M_SendData primitive also aggregates all responses as shown in Figure 9, therefore it returns 'negative confirm' to above EDTAMSE if it doesn't receive response from any participant until n(e.g., 3) consecutive retry.

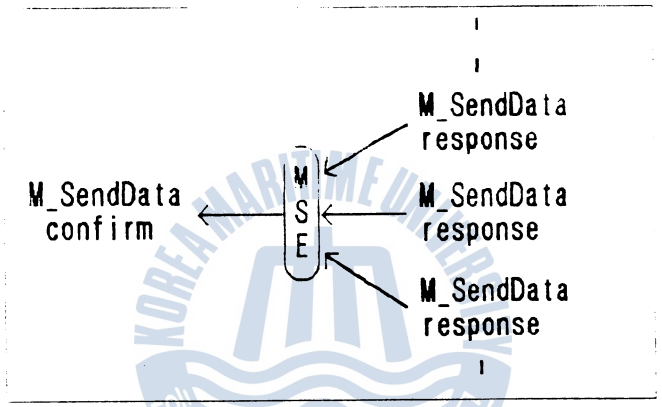


Figure 9. Aggregation of M_SendData response

Figure 10 shows an algorithm for multicast transmission. Unicast for any specific participant is also possible, and as many processes as group-work participants are forked in the case of multicast.

```
Algorithm for M_SendData primitive
input : group_id, class, member_name
output : return value(OK or NOTOK)
{
if (class == unicast)
  {
  unicast for member_name ;
  return OK ;
  }
else    /* multicast */
  {
```

```
initialize entry_number from group_id ;
fork as many child processes as entry_number ;
if (parent process)
  for( ; ; )    /* wait for responses from ALL child processes */
  {
  if (receive OK responses from ALL child processes)
    return OK ;
  else if (receive responses from ALL child processes, but there is NOTOK)
    return NOTOK ;
  else
      continue ;
  }
else   /* child process */
  {
  make MSD_PDU(PDU of M_SendData) ;
  call T_SendData primitive with MSD_PDU ;
  if (return value of T_SendData is OK)
    set the send state of this member to OK ; /* response to parent */
  else   /* result is NOTOK */
    {
    if (NOTOK for n(e.g., 3) consecutive T_SendData)
        set the send state of this member to NOTOK ; /* response */
    else
        set the send state of this member to OK ; /* response */
    }
   exit(0) ;
  }
 }
}
```

Figure 10. Algorithm for multicast transmission

5.3 Locking algorithm of EDTAMSE

Figure 11 shows a locking algorithm using D_Lock primitive of EDTAMSE.

```
Algorithm for Locking

initialize shared variable semaphoreID to 1 ;
initialize shared memory for lock-table ;
set timeout alarm ;

lock_req( )    /* process which invokes lock request */
input : object, member_id, group_id
output : return value(OK or NOTOK)
{
  for (i = 0 ; i < MAX_RETRY ; i++)
  {
    P(semaphoreID) ;
    check lock state of object in the lock-table ;
    if (object is LOCKED state)
    {
      V(semaphoreID) ;
      wait random time ;
      continue ;     /* retry until MAX_RETRY if object is already locked */
    }
    else    /* FREE state */
    {
      call D_LockRequest primitive with object, member_id & group_id ;
      if (return value of D_LockRequest is OK)
      {
        set lock state of object to LOCKED(by this member) ;
        V(semaphoreID) ;
        return OK ;
      }
      else if (return value of D_LockRequest is NOTOK)
      {
        set lock state of object to LOCKED(by other member).∴
```

```
            V(semaphoreID) ;
            wait random time ;
            continue ;
          }
          else    /* timeout alarm */
          {
            V(semaphoreID) ;
            wait random time ;
            reset timeout alarm ;
            continue ;
          }
        }
      }  /* end of for loop */
      print locking error message ;
      return NOTOK ;
    }
    lock_resp( )    /* process which receives lock request */
    input : object, member_id, group_id
    output : return value(OK or NOTOK)
    {
      P(semaphoreID) ;
      check lock state of object in the lock-table ;
      if (object is LOCKED state)
      {
        V(semaphoreID) ;
        call D_LockResponse primitive with NOTOK ;
        return OK ;
      }
      else    /* FREE state */
      {
        set lock state of object to LOCKED ;
        V(semaphoreID) ;
        call D_LockResponse primitive with OK ;
        return OK ;
      }
    }
```

Figure 11. Locking algorithm

When a participant tries to lock an object, a lock-table should be checked whether the object was locked by another participant or not, and send a lock-request to all participants if the object is in unlocked(free) state. Lock-request is multicasted to all participants through M_SendData primitive. For the case of system with multiple participant's processes, the lock-table is stored in shared memory among processes, and modification of table is synchronized by using semaphore. Furthermore, timeout mechanism is used for prevention of deadlock between lock-request and lock-response.

Above algorithm is similar to Ricart and Agrawala's optimal algorithm for mutual exclusion[16], but there are two differences as follows.

i) For multicast, Ricart and Agrawala's algorithm makes use of Send_Message routine repeatedly with for-loop, whereas above algorithm use MSE service in order to increase the degree of concurrent processing.

ii) A lock-request process in Ricart and Agrawala's algorithm waits until it enters critical section, whereas lock-request process in above algorithm reports error message if request fails until MAX_RETRY.

5.4 Evaluation and other applications

The GCP implementation is verified by tracing the association descriptor, primitives of service elements, and lock status of shared document.

```
A_AssociateRequest. indication
M_AssociateRequest. indication
ACCEPT -- 4
name : edtamd, socket : 4, invitor : 128. 134. 64. 1
called-AE AE-qualifier : edtam AP-id : 0 AE-id : 0
calling-AE AE-qualifier : edtam AP-id : 0 AE-id : 0
edtam : Connection request is arrived from mhkim of Group A
D_InitiateRequest from 128. 134. 64. 1 is accepted
M_SendData. indication
RO_Invoke. indication
Invoke-ID : 1
Lock request is arrived from mhkim
DOC-ID : 12 MODE : 0 OBJECT-TYPE : 1 OBJECT-ID : 345
LOCK M-ID : mhkim DOC-ID : 12 OBJECT-TYPE : 1 OBJECT-ID : 257 STATE : 2
LOCK M-ID : jhyim DOC-ID : 12 OBJECT-TYPE : 3 OBJECT-ID : 279 STATE : 2
LOCK M-ID : mhkim DOC-ID : 12 OBJECT-TYPE : 1 OBJECT-ID : 345 STATE : 2
                                -- ellipsis --
A_ReleaseRequest. indication
M_ReleaseRequest. indication
edtam : TerminateRequest is arrived from mhkim
```

Figure 12. Example of trace

Figure 12 shows an example of trace displayed on one system's console. In this figure, the value "4" is the association descriptor among systems, and associations among EDTAM AEs are initiated by a participant whose IP(Internet Protocol) address is "128.134.64.1". The progress of D_Initiate through A_Associate and M_Associate is also illustrated. After associations establishment, remote manipulation is invoked by RO_Invoke and M_SendData. If lock-request is issued, then lock-table entries are displayed.

In this paper, though the GCP is used for a collaborative document manipulation prototype system which provides remote manipulation about shared document replicas distributed in networks, other CSCW applications such as conferencing system, decision-supporting system also make use of the GCP. For example, let's consider the conferencing environment using the GCP as shown in Figure 13.

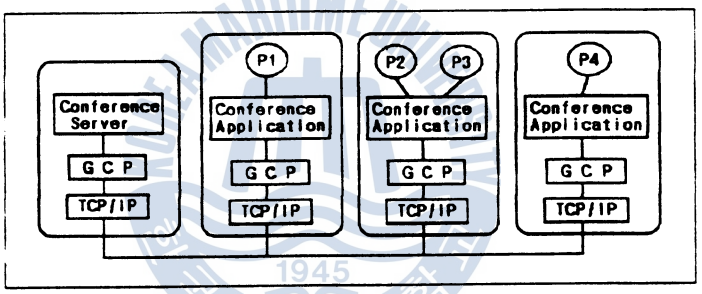


Figure 13. Conferencing environment

If we assume that conference server maintains original group-table and each conference application has group-table replica, then the following scenario is possible.

1) A participant P1 sends group-creation request message to conference server by using unicast of M_SendData.

2) Conference server changes group-table, and informs conference applications of that by using multicast of M_SendData.

3) Participants P2, P3 and P4 send join request messages to conference server.

4) Like step 2.

5) Conference applications establish associations among them by using M_Associate, and proceed with a conference by using M_SendData.

As in the case of above scenario, the dynamic group membership may be used, but server maintaining membership list is needed.

6. Conclusions

This paper describes a design and implementation of the GCP(Group Communication Platform) within the OSI application layer for supporing the cooperative work among multiple users geograpically distributed in networks. For maintaining the consistency of shared objects processed by the cooperative work participants, the GCP is designed and implemented to provide multicast service and lock mechanism. The GCP consists of existing OSI ASEs such as ACSE and ROSE, and new ASE, called MSE(Multicasting Service Element), providing multicast services. The GCP is tested as cmmunication protocols for a collaborative document manipulation prototype system using EDTAMSE(Extended DTAMSE).

The major advantages of the GCP are that the modification of operating system kernel is not needed and the GCP can be used on existing system providing point-to-point communication service.

References

1. Navarro, L, Prinz, W and Rodden, T, 'CSCW requires open systems', *Computer Communications*, Vol.16, No.5(May 1993), pp.288－297.
2. Vonderweidt, G, Robinson, J, Toulson, C, Mastronardi, J, Rubinov, E and Prasada, B, 'A Multipoint Communication Service for Interactive Applications', *IEEE Trans. on Communication*, Vol.39, No.12(December 1991), pp.1875－1885.
3. Yamaguchi, S, Shimojo, S and Miyahara, H, 'Implementation of Multicast Remote Procedure Call', *Proc. JTC-CSCC '88*(1988), pp.85－90.
4. Clark, W.J, 'Multipoint Multimedia Conferencing', *IEEE Comm. Magazine*, Vol.30, No.5(May 1992), pp.44－50.
5. Crowcroft, J and Paliwoda, K, 'A Multicast Transport Protocol', *Proc. SIGCOMM '88*(1988), pp.247－256.
6. Navaratnam, S, Chanson, S and Neufeld, G, 'Reliable Group Communication in Distributed Systems', *8th Int. Conf. Distributed Computing Systems*(1988), pp.439－446.
7. Halsall, F, *Data Communications, Computer Networks and Open Systems*, Addison-Wesley, USA(1992).
8. CCITT Recommendation T.431, DTAM : Introduction and General Principles(1992).
9. CCITT Recommendation T.432, DTAM : Service Definition(1992).
10. CCITT Recommendation T.433, DTAM : Portocol Specification(1992).
11. CCITT Draft Recommendation T.435, DTAM : Abstract Service Definition and Procedures for Document Manipulation(1993).
12. CCITT Draft Recommendation T.436, DTAM : Protocol Specification for Document Manipulation(1993).

13. ISO 8649/CCITT Recommendation X.217, Service Definition for the Association Control Service Element (1988).
14. ISO 9072-1/CCITT Recommendation X.219, Remote Operations : Model, Notation and Service Definition (1988).
15. ITU-T Recommendation T.122, Multipoint Communication Service for Audiographics and Audiovisual Conferecing Service Definition(1993).
16. Ricart, G and Agrawala, A. K, 'An Optimal Algorithm for Mutual Exclusion in Computer Netwoks', *Comm. of ACM*, Vol.24, No.1(January 1981), pp.8-17.

Appendix

```
typedef struct deinfo {
        int service_class;                  /* bulk transfer or manipulation */
        struct telematic_req {
              int fun_unit;                 /* functional unit */
              struct telematic_req *next;
              } *telematic_require;         /* telematic requirement */
        Appl_CAP *appl_cap;                 /* application capability */
        int protocol_version;
        int dtamQOS;
        char *user_info;
        } DEInfo;

typedef struct meinfo {
        char *group_id;                     /* group-identifier */
        struct group_entry {                /* group member entry */
              char *member_name;            /* member name */
              char *member_addr;            /* member address */
              unsigned short member_port;       /* member prot number */
              unsigned short s_f;       /* success or fail */
              }   group_table[ ];
        } MEInfo;

typedef struct aeinfo }
        char *ap_title;             /* application process title */
```

```
        char *ae_qualifier;        /* application entity qualifier */
        int ap_id;                 /* application process id */
        int ae_id;                 /* application entity id */
        }   AEInfo;

typedef struct m_object }
        int document_id;     /* document identifier */
        int select_mode;     /* selection of one node or all node in the subtree */
        unsigned short object_type; /* object type(e.g., chapter, section, etc.) */
        int object_id;       /* object identifier */
        }   M_Object;
```