공동작업을 위한 어플리케이션 공유 공동작업 엔진의 설계 및 구현

오 주 병[†] · 김 진 석[†] · 김 혜 규[††]

요 약

본 논문에서는 지역적으로 분산되어 있는 사람들간에 공동작업을 가능하게 하는 어플리케이션 공유 공동작업 엔진(ACE : Application sharing Collaboration Engine)을 설계하고 구현하였다. 어플리케이션 공유는 한명 또는 그 이상의 사용자들이 개인 컴퓨터 시스템 상에서 실행되고 있는 전용 어플리케이션을 다른 사용자들과 함께 공유하도록 하는 기술이다. 우리는 실시간 어플리케이션 공유가 가능하도록 ASO(Application Sharing Object) 객체와 그 행위들을 정의하였고, 공동작업 엔진은 ASO 객체들을 이용하여 분산된 시스템간에 공유기능을 처리한다. ASO 객체는 활성화 객체(activateASO), 수정 객체(updateASO), 입력 객체(inputASO) 그리고 제어 객체(controlASO) 들로 구성되어 있다. ASO 행위는 키보드 입력과 마우스 클릭과 같은 특정 순간에 발생되는 이벤트와 뷰 이미지 같은 어떤 순간에 유지될 수 있는 상태들의 변화에 의하여 발생 된다. 구현된 어플리케이션 공유 공동작업 엔진은 데이타 회의 시스템, 원격 교육, 그리고 엔지니어를 위한 프로젝트 공동작업 등에 적용될 수 있다.

The Design and Implementation of ACE(Application sharing Collaboration Engine) for Collaboration Work

Ju Byoung Oh[†] · Jin Suk Kim[†] · Hye Kyu Kim[††]

ABSTRACT

In this paper, we have designed and implemented ACE(Application sharing Collaboration Engine) which is possible an application sharing to collaborate among peoples who are geographically dispersed. The application sharing is a technology whereby two or more users collaborate to share the output of single application running on one computer system to the other users, and to provide input to the applications. We defined ASO(Application Sharing Object) object and its behavior to share applications in real time and ACE processes a sharing using ASO object among the distributed systems. ASO is classified into activateASO, updateASO, inputASO, and controlASO. The each ASO's behavior involves both events which occur at specific moments such as keystrokes and mouse clicks and more persistent status which can be observed at any time such as the image on the screen. The implemented ACE can be applied to the data conferencing, distance education, and project collaboration for engineer in distributed environments.

† 정 회 원 : 한국전자통신연구원 정보공학연구실
†† 정 회 원 : 한국전자통신연구원 정보시스템연구부
논문접수 : 1997년 7월 23일, 심사완료 : 1997년 12월 19일

1. Introduction

People do interact in group or organization. Interaction between peoples implicates collaboration. A collaboration described the following: the process of shared creation-two or more individuals with complementary skills interacting to create a shared understanding that none had previously possessed or could have come to on their own[10]. The collaboration was possible only if people shared a workspace. If collaborating peoples are situated at the same place, the workspace is, for example, a room, a table or white board which they share. But as organization grows and becomes more geographically dispersed, it is difficult for people to collaborate. In this environment, an application sharing[3][6][7] will become one of the most important technology for collaboration.

In the case of affiliated location of people, the shared application will becomes a shared workspace for their collaboration. So two or more users can communicate and collaborate as a group in real time through an application sharing. An application sharing can possible peoples who are geographically separated share text, graphics, image and applications and other information.

Each person can provide input, update text, or graphics, and discuss a problem to solve details. When we do collaborate, the shared information is able to be frequently updated. To achieve more better performance in application sharing, we define ASO(application sharing object) and its behaviors. Also we developed ACE(Application-sharing Collaboration Engine) to interchange and presented the shared information in real time. The structure of this paper is as follows: In section 2, we introduce the related work. In section 3, a collaboration system component will be presented. In section 4, we introduce Application Sharing Component. In section 5, we define ASO and the behavior of ASO. In section 6, we implemented ACE. Finally, we present our conclusions in Section 7.

2. Related Work

CSCW(Computer-Supported Cooperative Work) is the study of how people work together using computer technology[1]. Typical topics include use of email, hypertext that includes awareness of the activities of other users, videoconferencing, chat systems, and real time shared applications, such as collaborative writing or drawing. CSCW is often divided into the domains of synchronous (or real time) work, which considers peoples who are working together at the same time (such as with videoconferencing), and asynchronous work, which considers people coordinating their efforts across longer periods of time (such as with email). An ACE(Application sharing Collaboration Engine) sharing application is the domain of synchronous. Also, CSCW discipline can be placed on a grid with the time dimension running on the X-axis and the place dimension running along the Y-axis. Four major divisions are noticed: same place at the same time, different time at the same place, same time at the different place, and different time at the different place. Our collaboration engine is a same time different place. The same time different place method is a popular field of study within CSCW, since it looks at how to assist in the business need of cross-distance meetings. Shared workspaces use the concept of WYSIWIS to allow people in geographically distributed locations to work together at the same time and see what other participants are doing. Video teleconferences allow participants to see and hear each othe. across great distances and give the user more of a feeling of being in the same meeting room as the other participants.

Sharing applications are recognized as a vital mechanism for supporting group activities more effectively and on a more widespread basis[7][11][12][13]. Sharing applications mean that, when application programs executes any input from participants, all executions results of the application are shared among all participants. One approach to application sharing is to

develop special-purpose (collaboration-aware) applications designed for multiple users[14]. In many cases, however, it would be better for participants to simply share existing single-user (collaboration-transparent) applications, which only support the interaction between a user and the system [7][12]. It was called the GUI sharing method. The next is the replicated collaboration aware method in which it has an instance of the tool under distributed environment, each has local interface, collaboration is accomplished by approaching the sharing task, and input/output conformity and driving conformity for application are guaranteed[11].

MMConf[12], LIZA, and GroupKit provide a function for implementing the replicated construction system, of which typical systems include Rapport[15], Diamond, etc. This method is advantageous in that implementation of speaking right control mechanism in the tools is possible, and interaction among multiple users is feasible at the same time. It is disadvantageous in that it becomes more complicated in maintaining sharing state and interaction state, its approaching control becomes difficult at the same time, and there may be problems in performance speed as the number of users is increased.

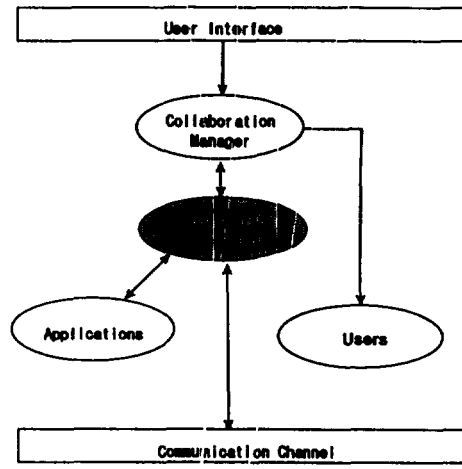
Monet, Rapport[15], Ceced, and Mermaid [2] presented the results of their study on sharing multimedia environment in the cooperative work field[2]. Audio file, which is the network transparent system with respect to distributed computer audio application using X window, provides abstract audio instrument interface with respect to simple network protocol[16]. The inter-application protocol based KWrite system of Apple MacIntosh system 7 has an open-type structure for multimedia documents which enable interactions among the users in the form of picture file for sharing documents. Gibbs, which has multimedia object concept applied which provides multimedia information delivery function for exterior entities such as screen or network, suggests the prototyping method of multimedia application and adopts component-directed view. There are also Standalone network-based video information capture as well as VidBoard system processing external instruments equipped with capture of television screen and transmission capacity[17]. In a GUI sharing, a single copy of the shared application runs at one site[7]. In a replicated method, a copy of the shared applications runs locally at each site[11][12]. The first is the centralized collaboration aware method in which multi-party software instance is shared by multiple parties by driving various interfaces. Its typical systems include systems such as Rendezvous [16] and Weasel, of which advantage is that interaction among multiple users is feasible by having the tool implement speaking right control mechanism internally. It is disadvantageous in terms of network traffic since it becomes more complicated under heterogeneous distributed environment because of hardware-dependent characteristics and all events are processed through the central system. For the replicated method, a distributed multiparty desktop conferencing system, called MERMAID, was developed for supporting a wide range of group cooperative work through effective integration of communication technology and information processing technology[7][8][9].

3. A Collaboration System Component

A collaboration involves multiple participants with shared goals and requires reciprocal exchange of ideas and extensive information sharing [10]. We define the collaboration system in (Fig. 1), which consists of user interface, collaboration manager, application sharing, and communication channel component. The collaboration manager component supports functions for establishing and joining a conference. The user selects the conference to join from a list of existing conferences, or create a new one by giving it a new name. It has a role of maintaining and managing information on all sessions occurring in connection with multimedia conferences which are progressed on networks. It manages session information and session status of

each session. It has a role of connecting and disconnecting a session among users at the time of collaboration through multimedia application sharing. The collaboration manager broadcasts the names of participants, conference titles and conference subject for all participants who wish to share multimedia application. When a conference is begun by sharing multimedia application, information on all users who are in the conference is shown in its collaboration manager. Then a user selects a conference by choosing one of them and constructs a session.

The application sharing component is a process whereby two or more user collaborate to share the output of applications running on one or more computer systems to the other systems and to provide input to the applications[7]. The application sharing component provides methods for sharing and unsharing the application. The communication channel component transfers the application sharing messages between peer application sharing components. Each conference for collaboration that is created automatically gets all default channels for a normal conference. A communication channel component has a function of multicast to group which is all participants to share application. A multicasting is a powerful paradigm for the development of the distributed applications and for CSCW groupware and multimedia applications. A multicast group is identified by the logical name assigned to it when the group is created. Each message targeted to the group's logical name is guaranteed to be delivered to all the currently connected and operational group's members Group communication systems,(e.g., Hours and totem, Ensemble, ISIS, Phoenix) provide group multicast and membership services. And data information and control information are differentiated and exchanged when information on application sharing is exchanged between the host and a server through networks. An application can include all tools(e.g. editor, drawer, OMT and compiler etc.) A user is any potential participant in a conference.



(Fig. 1) Collaboration based system

4. Application Sharing Component

The application sharing component enables multipoint application sharing by allowing a view onto an application executing, at one site to be advertised to other sites[2, 3, 6, 7]. Each site can, under certain conditions, take control of the shared application by sending remote keyboard and pointing device information. An application sharing component implemented as an ACE(Application-sharing Collaboration Engine).

ACE interacts with the collaboration manager component and the communication channel component to implement application sharing. An information is interchanged among ACEs in each sites using ASO message when events occur and/or status changes. Within an application sharing, windows are considered to be one of the following types[6].

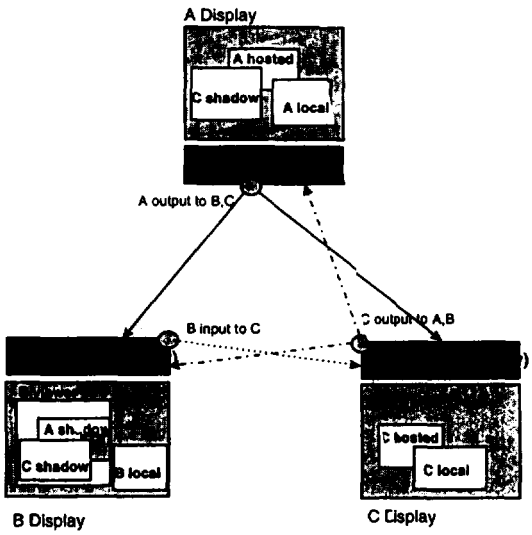
- Local: Local windows are not shared, that is, their output are only visible on the local system and the window can be controlled from the local system.
- Hosted: Hosted windows are owned by an application executing on the local system and are shared among another systems. The window's contents are replicated to peer ACEs and the window can be

controlled by ASO supplied by peer ACEs.

- Shadow: Shadow windows are drawn by the ACE and correspond to a hosted window on a particular peer ACE. The window's contents are rendered by the ACE using information supplied via the ASO message by the peer ACE on the hosting display. An input event directed to the shadow window is redirected to the peer ACE on the hosting system via ASO message.

(Fig. 2) shows an example of an application sharing. ACE A is viewing and hosting, it can both shares windows and displays shadow windows shared from other ACEs.

ACE A is managing a shadow window corresponding to the hosted window on ACE C, a hosted window which is shared. ACE B is viewing only, it displays shadow windows shared from other ACEs but does not itself share windows. ACE B is also managing two windows: a shadow window corresponding to the hosted window on ACE A, a shadow window corresponding to the hosted window on ACE C. ACE C is hosting only, it shares windows into the application sharing, but does not display shadow windows. ACE C is managing one window: a hosted window which is shared.



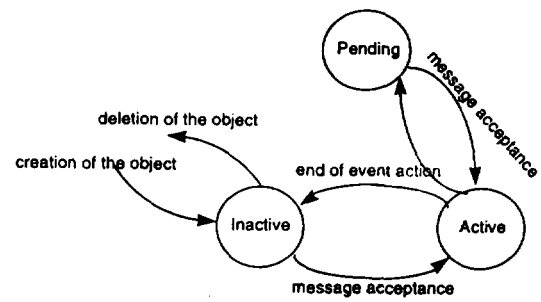
(Fig. 2) Example of an application sharing

5. ASO: Application Sharing Object

In ACE, an event action is performed by the event processor which is composed of ASOs. An ASOs become active when they receive messages. Each ASO in ACE has its local event action processor and it may have its local state memory, the content of which can be accessed only by itself objects. The state of an ASOs at a given time is defined as the contents of its local memory at that time. Upon receiving a message, an object executes a following actions: 1) creating of objects 2) referencing and updating of the contents of its local memory 3) acting event.

5.1 ASO object modes

An ASO object is always in one of three modes such as (Fig. 3): inactive, pending, or active[6]. An ASO is initially inactive. An ASO becomes active when it receives the message that defined in the specific messages. An active object can perform event action in ACE, and present the same result among ACEs. When an active object completes all actions that are performed in response to an accepted message, if no subsequent messages have arrived yet, it becomes pending state. An object in active mode sometimes needs to stop its current execution in order to wait for a message with specific message to arrive[9]. In



(Fig. 3) ASO object modes

such a case, an active object changes into pending mode. An object in pending mode becomes active again when it receives a required message.

Each object is assumed to have a conceptually queue for storing messages in the order of arrival[4]. These messages wait in the queue to be proceed by the object. The queue is considered to exist outside the object so that messages arrives regardless of the operation being currently performed inside the objects.

5.2 Message passing

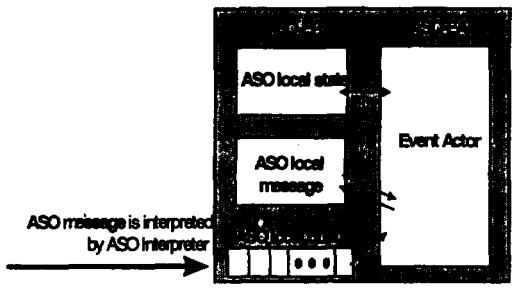
The general characteristics of message passing[4][[9] among in ASOs in ACE are given as follows.

- Broadcasting: An ActiveASO can send a broadcast message to all ASOs messages.
- Asynchronous: When an ASO sends a message to an another ASO, one ASO can send the message anytime, irrespective of the current state and modes of ASO
- Guaranteed Arrival and Buffered Communication: A message sent by an object always arrives at the destination within a finite time, and arrived messages are stored in a queue associated with the object.

5.3 The internal structure of an ASO Object

An ASO object consists of ASO-EP(Event Processor) and ASO-EA(Event Actor). Since each object has an event processor, it executes an event action one at a time in event actor. Although messages can be received by an object which is in active mode, the execution of the event action for the messages must be postponed until the current event action execution completes. Therefore each object has a message queue to store incoming ASO messages. (Fig. 5) illustrates the definition of ASO.

A member function such as AddTail( ), RemoveHead( ), GetHead( ), IsEmpty( ) and ClearList( ) is the function which is concerned by queue. Also a member function Put( ) writes data the socket. Get( ) read the data from the socket. A parameter is follow-



(Fig. 4) ASO object structure

```
Class ASO {
//Member Variables on;
  queue := a-message-queue;
  ASO state : = a-state-object;
  messageset:= a-set-of-messages ;
  event actor:= an-evaluator-object;

//Member Functions on;
//initialize;
  create;
  destroy;

//message arrival and reception through queue
AddTail(int flag, PACKETHEADER header, char
        *pData)
RemoveHead(int flag, LPPACKETHEADER pHeader)
GetHead(int flag, LPPACKETHEADER pHeader)
IsEmpty(int flag)
ClearList(void)

//Get message from local queue or put message to local
//queue. if mode is inactivc or pending then mode is
//active;

Get(LPINT pnType, LPINT pnDescript, LONG *plLen,
    char *pData)
Put(SESSION Sid, int nType, int nDescript, LONG lLen,
    LPVOID pData)

// interpreter and event actor execution on
// interpreter acceptance message to local message;
// if matched message found execute event action;
// if queue is empty then terminate of the
// execution;
Interpreter(int nMsg)
activateASO( );
inputASO( );
updateASO( );
controlASO( );
};
```

(Fig. 5) ASO object

ing. NType is the type of sending data. NDescript is the descript of sending data. LLen is the length of sending data. PData is the point of sending data.

(Fig. 6) is an algorithm of interpreter. An interpreter classify message which is received another sites. After message interpreter, if interpreter found matched message, send the message call to the proper ASO such as activateASO, inputASO. controlASO, updateASO., and execute event action. A each ASO object encapsulates member variables and the member functionson the data. The ASO can be classified in four categories objects as the following; activate ASO, update ASO, input ASO and control ASO.

```
Interpreter( int nMsg )
{
switch ( nMsg ) {
    case WS_READERROR : {...};
    case WS_WRITEERROR :{... };

This is activateASO message
    case WS_BROADCASTERROR :{ };
    case WS_READYTOACCEPT :
         AcceptClient();// accept client's connection.
         break;
    case WS_CONNECTED :
         break;
    case WS_SESSIONID_DATA :
         {   nGetSessionIDData( &nID );
             NetworkConnected();   }
         break;
     case WS_LOSTCONNECTION :
          NetworkCloseSession( );
          break;
     case WS_CLIENTNAME_DATA : { ... } break;

This is controlASO message
      case WS_USERDEF_DATA :
      // Get control data defined by user.
        GetUserDefineData();
        break;
        case WS_CONTROL_DATA :
       // Get control data
         GetControlMsg();
         break;
This is updateASO message
      case WS_SCRBMP_ARRIVED :
      // receive screen bitmap from server.
      break;
      case WS_SYSPALETTE_DATA :
      // system palette data arrived.
      GetPaletteData();
      break;
      case WS_SCRBMP_DATA :
      break;
      case WS_DIBCELL_DATA :
      // Update the received DIB cell to Original DIB.
       GetDIBCell( &DIBCell );
      case WS_VALUE_DATA :
       GetValueData();
       break;
This is inputASO
      case WS_WNDMSG_DATA :
      // Process remote control message
      GetWndMsgData();
      break;
This is extra message
     case WS_OVERTRAFFIC :
     MessageBeep( 0 );
     default :
     DefWsockProc( nMsg );
     break;
          }
}
```

(Fig. 6) Interpreter

• activateASO

This object is used for the activation of ACE. The ACE that wishes to become active shall send either a DemandActive or RequestActive to all ACEs within the conference. An ACE should initially send a RequestActive to determine if there are other active ACEs in the conference. If no active ACEs respond to the RequestActive, then the ACEs should subsequently send a DemandActive when it wishes to start sharing.

On receipt of a DemandActive, an ACE may send ConfirmActive to all ACEs within the conference. An inactive ACEs is not required to response to a DemandActive, it may choose to remain inactive. If an inactive ACE responded to aDemandActive, then it enters the active state. If an ACE is already active, then on receipt a requestActive, an ACE shall send ConfirmActive to all ACEs within the conference.

(Fig. 7) illustrates the use of RequestActive, DemandActive and ConfirmActive in the activation of ACEs, where initially no ACEs are active and ACE1 uses RequestActive to determine the activation state of other ACEs within the conference. ACE1 then starts sharing and uses DemandActive to initiate the activation of the other ACEs within the conference.

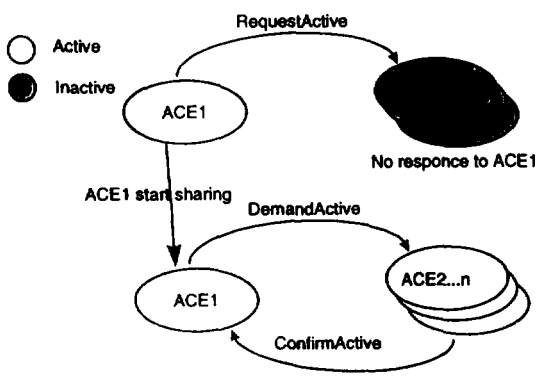
To deactivate another ACE, an ACE shall send a DeactivateOther. On receipt of a DeactivateOther, an ACE shall become inactive.

• UpdateASO

UpdateASO is the object that results in rendering to shadow windows. An ACE sends bitmap update to all ACEs within the conference by sending an UpdateBitmap message containing bitmap data. An ACE shall send uncompressed bitmap data or compressed bitmap data. On receipt of an UpdateBitmap(or palette) containing bitmap data, the receiving UpdataASO perform a copy from source bitmap data to destination window.

• InputASO

InputASO is the object that provides input to hosted windows. An ACE may provide input to peer ACEs within the conference by sending an Input message containing input events. Input events may be of one of the following two type:pointing device event or keyboard event.



(Fig. 7) The flow of ActivateASO

• ControlASO

ControlASO is considered to provide floor control. Floor control is a major determinant of application sharing usability. ControlASO is based on managing the rights to provide input to hosted and/or shadow window. When an ACE wishes to obtain the floor control, it shall send a RequestControl message to all ACEs. On receipt of a RequestControl message, the ACE holding the floor control shall send a GrantControl message with the current floor control. (Fig. 8) shows the ASO hierarchy. An activeASO, inputASO, ControlASO and UpdateASO is inherited by ASO object.

(Fig 8) is the type definition which is used by ASO object. We define some typedef for sharing application. It is a PACKETHEADER, WndMsg, DIBCELL, and SESSIONINFO. All messages include PACKETHEADER for changing information. A WNDMSG is a defined message which is used for sending of input information such as a mouse or a keyboard for remote control. In continuous transmission, we send cell which is changed. DIBCELL is the structure of cell information. SESSIONINFO is a session information of participant. This is used for the list of session participants.

```
typedef struct tagPACKETHEADER {
SESSIONID   FromSid ; //The session ID of   packet sender
SESSIONID   ToSid ;    // The session ID of packet receiver
int nType ;              // Type   The type of packet
int nDescript ;          // Packet description
long lLen ;              // Packet lengh
} PACKETHEADER

typedef struct tagWndMsg {
UINT uMsgType ;      // The type of message
WPARAM wParam ;  // wParam parameter of message
LPARAM lParam ;    // lParam parameter of message
} WNDMSG

typedef struct tagDIBCELL {
int nStartX, nStartY ; // The start point of DIB Cell
int nLenX, nLenY ;  // The length of DIB Cell
int nLineBytes ;       // The bytes per line of   DIB Cell
buf[CELLHEIGHT*CELLWIDTH*MAXPIXELSIZE]
; The content of real DIB cell
} DIBCELL

typedef struct tagSESSIONINFO {
SESSIONID SID ;  // The session ID of participant
int nMode ;     // The current mode of sessionparticipant
SOCKET DataSocket ; // The socket handle for sending
                                    data
```

```
SOCKET CtrlSocket ; // The socket handle for sending
                        control
char szAddress[ADDRESSSIZE] ; // The address of
                        Sessionparticipant
char szAlias[NAMESIZE] ; // Sessionparticipant name
char szHostName[NAMESIZE] ; // Host name of session
} SESSIONINFO;
```

(Fig. 8) The ASO type definition

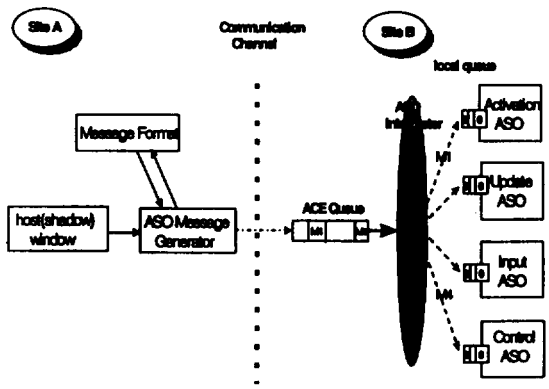
(Fig. 9) explains some API which is implemented in our system. This is a dynamic link library. These API are important functions which are necessary for implementing an application sharing system. The developers can implement an application sharing system using it. We don't explain about these function because the name implys theirs function.

```
ProcessMouseRemoteControl(WNDMSG WndMsg,
                        RECT rcScrSize)
ProcessKeyboardRemoteControl( WNDMSG WndMsg )
GetMouseInfoSize( HCURSOR hCursor, LPDWORD
                lpdwMask, LPDWORD lpdwColor )
DisplayCapturedBMP(pDC->m_hDC, &rcClient,
                CompareClient, DibNew);
BitmapCapture( &nLeft, &nTop, &nRight, &nWidth,
        &nHeight, &wBitsPerPixel,&bCompressedFlag,
        lpBitmapData );
ShowScreenImageData( HWND hWnd, HDC hDC,
                HDIB   hDIB, RECT rcUpdateDIB )
ReqRemoteControl( WPARAM wParam, LPARAM
                                lParam )
Send_BitmapMsg(char *Msg, int size)
Send_UserJoinMsg(char *Msg, int size)
SendServerScreen(SESSIONID SID, RECT rect)
SendDIBitmap(SESSIONID Sid, int nDescript, HDIB
                hDib)
SendSystemPalette(SESSIONID sid)
SendInitServerScreen(SESSIONID Sid, RECT rect)
int nGetDIBScanLineBytes(HDIB hDib)
SendWndMsg(SESSIONID Sid, UINT uMsgType,
                WPARAM wParam, LPARAM lParam)
GetServerSysPal(LPLOGPAL pal)
SendDIBitmap(SESSIONID Sid, int nDescript, HDIB
                                hDib)
SendSystemPalette(SESSIONID Sid)
SendInitServerScreen(SESSIONID Sid, RECT rect)
GetDIBScanLineBytes(HDIB hDib)
SendWndMsg(SESSIONID Sid, UINT uMsgType,
                WPARAM wParam, LPARAM lParam)
SendValue(SESSIONID Sid, int nDescript, int nValue)
SendString(SESSIONID Sid, int nDescript, char *str)
GetServerSysPal(LPLOGPAL pal)
SendNextServerScreen(SESSIONID Sid, RECT rect)
SendNextCell(THREADINFO *pThreadInfo)
CompareCell(HDIB hDibOld, HDIB hDib, int sx, int sy,
                                int xLen, int yLen)
SendDIBCell(SESSIONID Sid, HDIB hDib, int sx, int sy,
int xLen, int yLen, int nDescript)
GetDIBCell(DIBCELL *pDibCell)
UpdateDIB(LPDIBCELL pDibCell)
GetModifiedDIB(void)
InitRect(LPDIBCELL pCell)
GetSystemLogPal(LPLOGPAL pLogPal)
```

(Fig. 9) The implemented major API for ASO

5.4 The scheme of ASO object processing

In this section, we present a scheme of ASO object processing(see (Fig. 10)). When user inputs events, an ASO message may be created by ASO message generator and it is interchange between sites.



(Fig. 10) The scheme of ASO object processing

ASO become active when they receive messages, and an event processing proceeds as ASO message transmissions among objects. An ASO messages are the unit of information to be interchange and a coded representation of an instance in application sharing.

The ASO messages are defined only at the interchange point between the active site A and site B. When site A wishes to send an ASO message to site

B, it calls an ASO message generator which converts the internal format used by ACE. When site B receives an ASO message, the object is decoded by an ASO interpreter. The value within the object are passed by the ASO interpreter which may convert them to the internal format used by ACE.

5.5 The behavior of ASO objects

An ASO behavior involves both events which occur at specific moments(e.g. keystrokes and mouse clicks) and more persistent status phenomena which can be observed at any time(the image on the screen). Events are atomic, non-persistent occurrences in the windows. As the mouse cursor moves across the boundary of a window, that window can be activated as the focus. Events can trigger status changes.

- Telepointing: The telepointing[1, 2, 3] provides a user with a set of globally visible pointers. When a telepointer is moved into a shared window, it becomes visible to all participants. Therefore, The telepointer behavior tracks the movements of locally used telepointers and distributes these movements to the remote sites.

- Control Behavior: A control assignment policy must be set down so that only one participant has the floor at a time during a conference. The floor user has the right to provide input to the application being shared. There is a distinction between floor policies and floor mechanisms. Floor policies describe how participants request the floor and how it is assigned and release.

- Update Behavior: Updates to shared information objects may affect presentation. An ACE sends bitmap updates to all ACEs within the conference by sending an UpdateASO containing bitmap data.

- Remote Sharing Behavior: In some application sharing scenarios, it is useful for sharing of applications and/or windows to be initiated remotely, that is, applications running on the local terminal are shared not by a local end-user or programmatic action, but rather as a result of a request from a peer ACE.

- Dynamic Sharing Behavior: Dynamic sharing allows you to share an already running application to a new display. Sometimes this is also called accommodation of latecomers.

- Activation: This situation may occur at initial point of application sharing. An hosting ACE that wishes to active another ACEs shall send ActivateASO to all other ACEs.

5.6 Scenario and character of ACE

1) Creation of conference

A conference helding is clicked for initiating a conference. In case of helding a conference, a conference helding radio button is selected and the names of participants, conference title, conference subject, etc. are inputed. In case of participating in a conference, a conference participation button is selected, the names of participants are inputted, and the conference title of a conference to participate among the contents in the conference list is selected and clicked when there is a conference being held.

2) Establishment of transmission region

A conference holder may set the transmission region in its screens. The transmission region is set in the following methods: 1) Specific window 2) User define area 3) Entire screen. A specific window is used when one window among many application programs on its screen is selected and transmitted. A user define area method is used when a user wishes to transmit in an arbitrary size. A region to be transmitted is set by moving a mouse. An entire screen transmission method transmits an entire screen at present. Since a large amount of data are transmitted at this time, speed may be affected.

3) Screen transmission

Transmission can be begun when designation of a transmission region is completed. There are two forms of transmission as follows:

- Continuous transmission

  A screen is transmitted continuously. This is a mode of continuously transmitting the present screen to participants in the conference connected. It is selected to be in the basic mode at present.
- Non-continuous transmission

  This is a mode of transmitting the present screen when transmission initiation is selected and then waiting until the next transmission initiation is selected.
- Transmission starting

  After cstablishment of transmission region and transmission mode is completed, the image of a screen can be transmitted actually. Transmission is begun when transmission initiation is selected.
- Transmission stopping

  This menu is selected when temporary stopping of transmission is desired. Then transmission is stopped and a picture showing stopping of transmission appears on the screen.

4) The assignment of presentation

When a conference is held for the first time, a conference holder becomes a presentation manager. A presentation manager is a term referring to a conference participant who transmits its screen. It is possible to have another conference participant transmit the screen. This can be done by selecting presentation assignment in the menu. A conference holder selects a desirable participant. To the selected participant, a message box asking "Presentation assignment request" is received. Presentation assignment message box appears. If "Yes" is selected, the presentation manager is changed into a participant assigned to the conference holder and a picture indicating the change is shown as follows: The present presentation manager may be released forcefully. If the present presentation manager is another participant other than a conference holder, the conference holder may release the present presentation manager forcefully and become a presentation manager. "Presentation release" is selected in the menu.

5) Request and release of floor control

When controlling of application program(s) in a screen being transmitted by the present presentation manager is desired, floor control request may be made. Release of floor control may be done by the person who requested it or may be done forcefully by the presentation manager.

6. ACE engine

ACE(Application-sharing Collaboration Engine) enables application sharing by allowing a view onto a computer application executing at one site to be advertised within a conference to other sites. Each site can take control of the shared computer application by sending remote keyboard and pointing device information.

ACE can cooperate via the ASO messages to share one or more applications within the conference. ACE provides an interface to a collaboration manager component and to a communication channel component to implement application sharing. Data is exchange between peer ACE using ASO messages.

ACE consists of an event capture, ASO message generator, distributor, ASO message fetcher, ASO message interpreter and an event processor as shown in (Fig. 11) By being able to be converted from a server to a client and vice versa according to the type of work (existence of application), it implies both aspects of server and client.

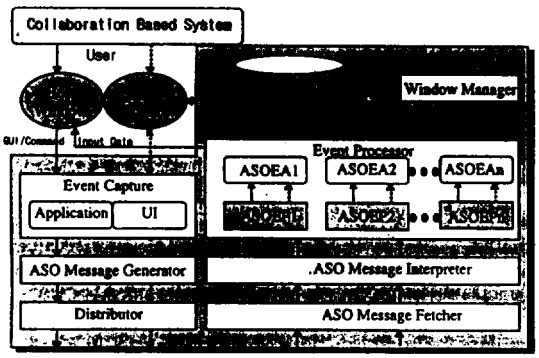
An event capture takes all output stream from the host windows or shadow windows. An output streams are segregated into two categories: bitmap data and input event. When an ACE is hosting a window, it is responsible for constructing an bitmap data for that window which will allow other ACEs to faithfully draw corresponding shadow window.

When an ACE is shadowing a window, it is responsible for sending an input event from shadow to host.

ASO message generator create ASO message to interchange among ACE. It have a ASO intermal message format and converts output stream into ASO internal message format used by ACE. All ASO messges are multicasted via the communication channel by a distributor.

ASO message fetcher takes ASO messages from the ACE queue, forwards them to the ASO message interpreter. ASO message interpreter decodes the ASO message. An event processor executes the event which is translated from the ASO message interpreter.

The event processor consists of four ASOs: ActivateASO, UpdateASO, InputASO and ControlASO, each of which is activated in accortance with the ASO message.



(Fig. 11) The architecture of ACE

- Event Capture

  This is a module capturing initial image state of sharing application in the Host and Shadow, input event, and GUI image with respect to input event processing.

- ASO Message Generator

  This is a module classifying the events captured in the event capture and producing ASO Messages for each event.

- Distributor

  This is a module assuming a role of distributing messages made in the ASO Message Generator to the sites of all developers.

- ASO Message Fetcher

  This is a module assuming a role of bringing each message delivered from the Host or Shadow in the order of storage one by one.

- ASO Message interpreter

  ASO Message interpreter decodes the ASO Mess age which is created by the ASO message generator.

- Event processor

  An event processor executes the event which is translated from the ASO message interpreter. It uses the presentation library and window manager.

- Presentation library

  This is composed of a dynamic library related to a window for displaying the same result as the application performance of the Host side.

- Window manager

  This is a portion defining the structure of identifiers of the windows being performed at the corresponding site. It has the mapping information on the corresponding ID being performed of the Host in the delivered message of the Host and on ID to be performed at the Shadow.

7. Conclusion

We designed and implemented ACE(Application sharing Collaboration Engine) to share application in real time for collaboration work. Application sharing is divided into GUI sharing and replicated sharing. Our collaboration engine supports GUI sharing to collaborate in distributed environment. We have defined ASO(Application Sharing Object) object and its behaviors to share application to real time. ASO is an unit of information to be interchanged and a coded representation for application sharing. We have classified ASO into four objects: activateASO, updateASO, input ASO and controlASO. This approach confines ASO's behaviors to within the object and allows the action to handle the object. ACE processing ASO ob-

ject in distributed environment consists of event capture, ASO message generator, distributor, ASO message fetcher, ASO message interpreter and event processor. We are currently implementing ACE based on T.120 for the interoperability in application sharing. At a conclusion, two or more users can communicate and collaborate as a group in real time through ACE processing ASO object.

Reference

[1] Walter Renhard, Jean Scweitaer, Gerd Volksen, and Michael Weber, "CSCW tools : Concepts and architecture", IEEE Computer, pp. 28-36, May 1994.

[2] K. Watabe et al. "Distributed Multi-party Desktop Conferencing System : MERMAID." Proc. CSCW 90, ACM Press. New York. 1990. pp. 27-38.

[3] K. Strinivas etc, "Monet : A Multimedia System for Conferencing and Application Sharing in Distributed System". Proc. First Workshop on Enabling Technologies for Concurrent Eng. enabling Technologies Group, CERC, WestVa, Univ., Morgantown, W. Va., 1992, pp. 21-37.

[4] Vittal, John, "Active Message Processing : Message as Messanges", in Computer Message Systems, R.P.Uhlg, editor, North-Hotland Publishing Company, 1981.

[5] ITU-T Recommendation T.120 (1996) Data Protocols for Multimedia Conferencing.

[6] ITU-T Recommendation T.Share (1996) Application Sharing.

[7] S. Greenberg, "Sharing views and interactions with single-user applications," Proc. Conf. On Office Information System, pp. 227-237, April 1990.

[8] G. Agha, "Actors : a model of concurrent computation in distributed system", MIT Press, 1986.

[9] A. Yonezawa, "ABCL : An object-oriented concurrent system", MIT Press, 1990.

[10] Hsinchun Chen, "Collaborative System", IEEE Computer, pp. 58-66, May, 1994.

[11] J. C. Lauwer,et al., "Replicated Architecture for Sharing Window System : A Critique," Proc. Conf. On Office Information System, pp. 249-260, April 1990.

[12] T. Crowley, et al., "Mmconf : An Infrastructure for Building Shared Multimedia Applications," Proc. Conf. On Computer Supported Cooperative Work, pp. 329-342, October 1990.

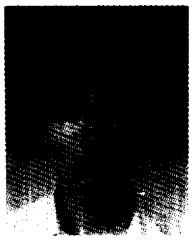
[13] S. R. Ahuja, J. R. Ensor, and S. E. Lucco, "A Comparison on Application Sharing Mechanisms in Real-Time Desktop Conferencing Systems," Proc. Conf. On Office InformationSystems, pp. 238-248, april 1990.

[14] M. D. P. Leland, R. S. Fish, and R. E. Kruat, "Collaborative Document Production Using QUILT," Proc. Conf. On Computer Supported Cooperative Work, pp. 206-215, September 1988.

[15] S. Ahuja, J. Ensor, and D. Horn, "The Rapport Multimedia Conference System", Proc. ACM Conference on Office Information System, 1988, pp. 1-8.

[16] T. Levergood, A. Payne, J. Gettys, G. Treese, and L, Stewart, "AudioFile : A Network Transparent System for Distributed Audio Application", Proc. Usenix Summer Conference, 1993, pp. 22-33.

[17] J. Adam and D. tenneenhouse, "The Vidboard : A Video Capture and Processing Peripheral for a Distributed Multimedia System", Proc. ACM Conference on Multimedia, 1993, pp. 113-120.



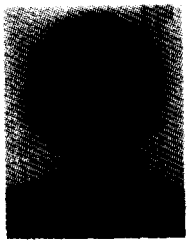
오 주 병

1992년 충남대학교 공과대학 컴퓨터공학과(학사)

1994년 충남대학교 대학원 컴퓨터공학과(석사)

1994년~현재 한국전자통신연구원 정보공학연구실 선임연구원

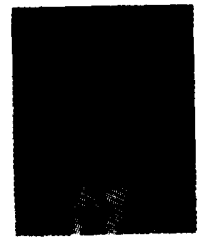
관심분야 : 디렉토리 서비스, 객체지향시스템, 멀티미디어, CSCW



김 진 석

1982년 울산대학교 전자계산학과 졸업(공학사)
1988년 동국대학교 전자계산학과 졸업(공학석사)
1992년 정보처리 기술사
1982년~현재 한국전자통신연구원 정보공학연구실 실장(책임연구원)

관심분야:CSCW, 멀티미디어 데이타베이스, 소프트웨어 공학



김 혜 규

1973년 서울대학교 공과대학 응용물리학과(학사)
1985년 서강대학교 경영대학원 경영학과(석사)
1994년 서강대학교 공공정책대학원 정보처리(석사)
1979년~현재 한국전자통신연구원, 정보시스템연구부장(책임연구원)

관심분야:정보산업정책, 멀티미디어