1-1-1992

EIES 2 : a distributed architecture for supporting group work

Computerized Conferencing & Communications Center

James Whitescarver

Robert M. Czech

Sreenivas Reddy

Ajaz R. Rana

See next page for additional authors

Authors
Computerized Conferencing & Communications Center, James Whitescarver, Robert M. Czech, Sreenivas Reddy, Ajaz R. Rana, and Murray Turoff

RR# 28

EIES 2 - A DISTRIBUTED ARCHITECTURE FOR SUPPORTING GROUP WORK

James Whitescarver

Robert M. Czech

Sreenivas Reddy

Ajaz R. Rana

Murray Turoff

DRAFT

January 1, 1992

New Jersey Institute of Technology

ABSTRACT

The Computerized Conferencing Center (CCCC) at New Jersey Institute of Technology (NJIT) has been researching on-line group communications for 17 years by developing and studying tools to advance the "collective intelligence". The Electronic Information Exchange System 2 (EIES2) provides a research, development and operational environment for distributed computer supported cooperative work (CSCW) systems. The EIES 2 distributed Smalltalk processor provides for rapid prototyping and implementation of muti-media CSCW facilities in the network environment. The Smalltalk support of the object model, and meta-language properties make it ideally suited for incremental development CSCW applications. The EIES2 communication environment supports a decentralized network architecture. Modern standards are used in the implementation of data structures, communication interfaces and database. The EIES2 application layer protocols support use ASN.1 data representation to access to an object-oriented distributed database via X.ROS remote operation services. EIES2 can serve as a foundation on which group work systems may be built and defines protocols that can allow them to inter-operate. An initial system presents a powerful metaphor of conferences and activities which provides an extensible framework upon which to add group work applications. Work to date has provided structures for information exchange, inquiry networking, information filtering, the on-line virtual classroom, and group decision support. This paper presents the system architecture model used for EIES2 and describes the implementation and current applications.

CONTENTS

1. An Architectural Model for Group Work

1.1 The Organization

A typical organization consists of individuals performing different functions, using various information and physical resources. The activities performed by the individuals, and the flow of information combine to generate activities of the organization. Activities in even the most highly structured organizations are subject to change. A top down structured definition of the activities has a very limited life expectancy. An analytical definition of all the activities and their relationships may be impractical or impossible. An alternative is synthetic definition of organizational activities allows individual activities to be defined in terms of the information objects they involve and the roles of individuals taking part in the process. Contrary to current wisdom in software engineering, when it comes to collaborative communications applications, a bottom up design process is more appropriate. At the same time, there should be a consistent organized framework within the on-line environment where all the various on-line activities may be handled with a common interface.

1.2 Network Environment

The participants in organizational processes may be widely distributed. Inter-organizational activities are even more distributed. Main-frame computers, departmental mini-computers and desk top personal computers must work together in a coordinated manner to provide a "connected" organizational support environment. Applications should access remote services as needed. The network should be transparent to the user.

Data objects may be anywhere on the network. Information must be shared between applications. Furthermore, a coordinated and distributed data base providing controlled access to organizational information is required.

1.3 The OBJECT Model

The object model supports synthetic definition of applications in terms of objects and messages they can receive. An object receives a message and performs a method based on the message contents and the class of the object. The definition of organizational activities in this way not only allows the activities to be defined independently, but also allows them to be changed independent of the rest of the system. Procedural definitions are not excluded by the use of the object model, thus, allowing both highly structured and flexible activity relationships to be defined. Inheritance is a property of object-oriented systems that allows new object classes to inherit functionality from existing classes. This greatly simplifies development in that it is only necessary to define how a new class differs from it's closest related existing object class.

1.4 Decoupled User Interface

As the group work environment evolves, the features and facilities will change as will the user interfaces. User interfaces may be as numerous as the different individual views of the organization. Users must not be subjected to unnecessary changes in the interface. A single consistent interface should provide access to all group work activities but the interface need not be the same for all user participating in a group process. The functions of the application and the user interface should be maintained independently. The object model will support such independent definitions of function and the user interface beautifully. This separation is essential to support the differing user views and the evolutionary nature of CSCW implementation. /* Ultimately the user drives the evolution of the CSCW environment. Individual productivity is increased when the computer performs tasks for the benefit of the individual. In an environment that supports decoupled user interface development, the user agent provides the mechanisms to perform the functions on the users behalf. In some cases, the task may be fully automated and eliminate the user interface entirely. Otherwise, the user agent provides the interface to the user for the desired functions. Advanced user programming facilities for the user agent are required At minimum, user agent programs should be

user readable[1] so that users may understand and validate that the functions acting on their behalf agree with their desires. */

1.5 Standards

Although standards are not the only way to solve inter-operability problems, standards are an effective means of minimizing inter-operability problems. Instead of building an interface to each pair of systems, inter-operability is enhanced if each system can support a standard interface.

ISO ASN.1 CCITT ASN.1[2] provides a notation and machine independent format for structured data to be exchanged between systems. This abstract syntax notation is then used to specify the protocol units for virtually all the new international application layer standards. X.ROS defines a protocol for remote operation services.

The support of these modern standards is a basic requirement for future office information systems. New standards are needed to fully support group work applications. The EIES2 protocols represent a general mechanism for operations on remote objects which allows the implementation of distributed group work applications[3].

2. EIES2 IMPLEMENTATION

2.1 EIES2 OVERVIEW

The Electronic Information Exchange System II (EIES2) provides an integrated collection of tools for defining and operating computer-supported cooperative work (CSCW) systems. Facilities are provided to tailor group communication activities and user interfaces in a network environment. Standard application layer protocols for remote operations are used to support a distributed database of communication objects in an arbitrary network of application entities (AEs).

A high-level tool, the Communications Language Processor (CLP), is provided to define the protocol data units (PDUs) for existing, and future standards. The CLP uses standard abstract syntax notation (ASN.1) as defined by the ISO and

CCITT. This standard has been applied to define virtually all the new application layer standards.

EIES2 employs the X.ROS application layer protocols for remote operations. The X.ROS remote operations protocol is also achieving a high degree of acceptance as a new application layer standard. The EIES2 protocols provide an extremely general base, employing the existing standards as much as is practical, upon which future CSCW standards may be built.

EIES2 provides its own distributed Smalltalk environment for CSCW application development. This maps well to the network environment of CSCW, and provides for a high degree of security. The object model is extremely well suited to CSCW development.

2.2 SYSTEM ARCHITECTURE

2.2.1 COMMUNICATIONS_PROCESSOR_(CP)

The Communications Processor is an object-oriented, stack-based bytecode pseudo-machine. The CP is modeled after Smalltalk-80, developed at XEROX PARC. It is an extended version of the Smalltalk bytecode interpreter defined in [4] (also known as "The Blue Book").

2.2.2 EIES2_NETWORK_INTERFACE

The Network Interface connects EIES2 compatible Application Entities (AE's) allowing distributed and decentralized processing. AE's may run on the same CP, and CPs may run as separate processes on the same machine or on separate machines, either tightly or loosely coupled. The type of connection is transparent to the application.

The communication between one or more AE's is handled by two systems, the Remote Operations Server (ROS) and the Reliable Transfer Server (RTS). These facilities are extremely important to the EIES2 distributed system architecture,

---

* Smalltalk-80 is a trademark of XEROX Corp.

since one process executed by an AE may invoke several remote messages, each of which would be executed by other remote AE's thus "distributing" the original task.

In addition to providing its own ROS, RTS, and support for UNIX System V interprocess communications, EIES2 interfaces to Northrop ISODE to provide common application service element (ISO 8850), presentation (ISO 8823), session (ISO 8327/CCITT X.225), and transport (ISO 8073/CCITT x.224) services over CCITT X.25 and ARPA TCP/IP networks.

2.2.3 EIES_2_DATA_MANAGEMENT_SYSTEM

The EIES II Data Management System (DMS) provides a structured virtual object memory for the Smalltalk bytecode interpreter used in the Communications Processor (CP). Provisions are made for operations performed on entire objects or on the internal structure of an object. Operations may be performed on objects in memory, on linear disk segments or on disk indexes. Data is stored and defined using the ASN.1 international standards. Object size is limited only by storage capacity.

The CP operates on both local and remote objects. If a remote object is to be used, the Remote Operations Server (ROS) is invoked.

2.2.4 OPERATING_ENVIRONMENT_INTERFACE

The operating environment interface provides an interface to the local system. It provides access to local files and devices, including the system clock, interprocess communications, semaphores, and operating system commands.

The initial environments supported are UNIX* compatible systems.

The operating environment interface is implemented using C library routines. The resulting EIES2 virtual operating environment interface isolates system dependencies from all

---

* Trademark of AT&T

other system components.

2.2.5 USER_INTERFACE_FACILITIES
The user interface connects the user to one or more application entities AE's. It provides facilities to enter, select, display and update EIES2 objects. It provides access to all functions of the communication processor by using the Smalltalk language.

The terminal is modeled as a multimedia device supporting many virtual interfaces. High-level interface functions allow the support of terminals with radically different physical characteristics.

The text facility provides simple document formatting (ODA+ is planned), full screen operations, and pointer-structured text operations. A dumb terminal interface provides a line-oriented interaction and provides complete terminal independence. The type of terminal used is transparent to the application program.

2.2.5.1 THE_FORMS_PROCESSOR_(FP) The ability to communicate with forms rather than simple text is extremely useful in the CSCW environment. A form represents an implementation of structured messages as described by Malone [5]. In the AMIGO Model the EIES2 form structure represents a Message Type in the context of Activities defined for group communications . A form can also be "attached" to a message or to a conference comment. When the user views the incoming message or the comment, he also will be prompted by the system to fill out the form. When the user completes the form, the information carried by the form will be returned to the sender or other activity agent.

2.2.5.2 MENUS The Menu facility provides for horizontal or vertical pop-up menus. Items may be selected by number, letter, and by cursor control.

2.2.5.3 WINDOWS The window sub-system provides device independent support for terminal input and output. The Window subsystem also provides multiple pop-up windows and maintains an answer- ahead buffer. Controls indicating exception situations during terminal I/O operations are maintained and can be accessed by any upper level subsystems. These controls interface with the help

processor, the command processor, and the CP to support consistent input processing.

2.2.6 EIES2_DISTRIBUTED_SMALLTALK

The communication processor is data-driven, based on the database object and method definitions. The processing done by a particular CP is determined solely by the messages received by the objects local to that CP. Network load can be balanced simply by migrating objects between CPs. EIES2 provides its own distributed Smalltalk* environment for CSCW application development. Smalltalk provides a message-based architecture that is highly suited to the CSCW environment. In the Smalltalk object model, objects receive messages and perform methods based on the class of the receiver. This conforms well to the network environment of CSCW, and additionally provides a high degree of security. /* In addition, Smalltalk has excellent meta-language properties. Object classes and methods define the statements that are valid for the objects. As higher-level capabilities are defined, so are higher level statements. This effectively elevates the language from the level of the machine to the level of the application. */ The environment is well suited to rapid prototyping of CSCW interfaces, implementation of new applications, and provides for end user and group tailoring of the CSCW environment. User agent programming provides for the automation of repetitive user functions. Neither the user nor the application developer need be aware of where the object is located on the network.

3. EIES2 SMALLTALK OVERVIEW

Just like functional and logical languages were associated with the development of a particular language (like LISP and PROLOG respectfully), Object oriented programming owes a lot to the development of Smalltalk. Like some of the macro languages which were associated with particular constructs (LISP was associated with lists, TRAC with strings and FORTH with words) Smalltalk is associated with Objects (actually

---

* Trademark of Xerox Corp.

along with messages). The single construct in the Smalltalk language is the message send-- sending a message to an object. Smalltalk belongs to a small class of single construct languages. These languages support a high degree of programming leverage. Enforcement of the object model and superior meta-language capabilities make it ideal for evolutionary programming environments.

Anything in Smalltalk is an object and this ranges from an integer constant to a computational process. Smalltalk is a simple language, but it uses unconventional terminology and it differs conceptually from conventional programming languages.

In the Smalltalk object model, objects receive messages and perform methods based on the class of the receiver.

For example:

```
object message.
```

is a legal Smalltalk statement which sends the message, "message", to the object, "object".

In the above example, the class definition for the class that "object" belongs to, must have the message, "message" defined, or an error, "message not understood" results. The set of messages that an object class understands is called the "protocol" for the class.

This model supports incremental software development with excellent data security and protection. Systems are developed by defining the behavior of objects independent from the rest of the system. Each object class definition completely defines how the objects of that class can be "legally" manipulated. There is no mechanism in Smalltalk to access objects except by methods defined for its class.

3.1 Meta Language Properties

Smalltalk has excellent meta-language properties. Object classes and methods define the language statements that are valid for the objects. As higher-level capabilities are defined, so are higher level statements. This effectively

elevates the language from the level of the machine to the level of the application.

Although Smalltalk itself, is a very simple language, each Smalltalk application defines a specialized very high level language tailored for that application. The capabilities thus defined are added to the arsenal of functionality available to the system developer in the Smalltalk environment.

Our sample statement "object message." becomes part of the language if and only if we define that message for the object. The definition of application oriented messages defines the language of the application-- Smalltalk style.

3.2 Conceptual Programming in Smalltalk

Smalltalk methods operate on "untyped" data. This means that the programmer need not be aware of what kinds of data the methods will be used to manipulate in the future. The methods developed define "conceptual" algorithms that can manipulate any kind of data. This further enhances the "tool building" properties of the language.

A sort algorithm, for example, can sort any kind of objects, as long as the messages send the objects are supported, e.g. compare messages.

3.3 Leverage in Smalltalk

Smalltalk object classes are defined in a hierarchy that allows objects to inherit properties from existing classes. This allows programmers to define new object classes by only specifying how they differ from similar existing classes rather than defining all the properties of object classes from scratch.

The hierarchy of classes, is defined by specifying the "super class" in each new class definition. The new class inherits all the variables and messages defined for the super class.

3.4 Distributed Operation

Most Smalltalk environments have been used for individual rather than group applications. They have a separate database of objects for each user and cannot be easily applied to group support applications. The database is "transient" memory in main memory that only becomes permanent if the memory "image" is stored on disk for later reuse.

The EIES2 distributed Smalltalk takes advantage of the message base model by supporting a distributed database of persistent objects shared among all communicating EIES2 processes. Messages to remote objects are literally sent across the network by remote operations (X.410).

The programmer does not need to be aware of where objects are on the network. The same program will work regardless of the location of the data.

A technique called "lazy execution" maximizes the overlap (parallelism) of processing over the network to minimize response time. Because Smalltalk methods represent high level application specific operations, communications over the network is minimal compared with remote data access or remote window management protocols. The distribution of processing is controlled by the location of the objects.

This environment has to potential to support a virtually unlimited size user community simply by adding processors to the network.

3.5 OBJECT CLASSES FOR CSCW

3.5.1 Communication_Objects

In addition to the standard Smalltalk-80 classes and methods, EIES2 provides support for the Class CommunicationObject to support CSCW. A communication object is responsible for controlling access to objects in a network environment. The protocol includes methods to assign and test roles in the hierarchy of communication objects. Subclasses include User, View, System, Item, and all other classes of objects accessed via loosely coupled links on the CSCW network.

3.5.2 View

A View represents a very general class of objects for CSCW. In the simple case, a View is a list of one or more communication objects. Views also support waiting versus accepted items, memberships, indexing and sub-views. The protocol for Views includes methods for submission and delivery of communication objects, changing membership and roles, accessing waiting or accepted objects, and obtaining display information in an one line list format, four line scan format, or full view format. Views are used to support mailboxes, conferences, queues, etc. and to control access to all Items on EIES2.

3.5.3 Item

An item is itself a sub-class of View, since it may have a list of replies or sub-items associated with it. Items contain Texts and optional Attachments.

3.5.4 UserAgent

An object in the UserAgent class provides a user interface to the functions of the CSCW environment. It contains a set of method blocks corresponding to each state of the user agent as well as the current state and context of the user. Methods support standardized Terminal and Window facilities, trap handling, state changing, and application specific functions.

3.5.5 System_Classes

The RemoteObject class supports operation across user, database, application, and network boundaries. The Result class support lazy process execution to maximize concurrency and provide additional concurrency control. ASNStream and ASNString classes support ASN.1 object classes for standards support.

3.6 EIES2 Smalltalk Example

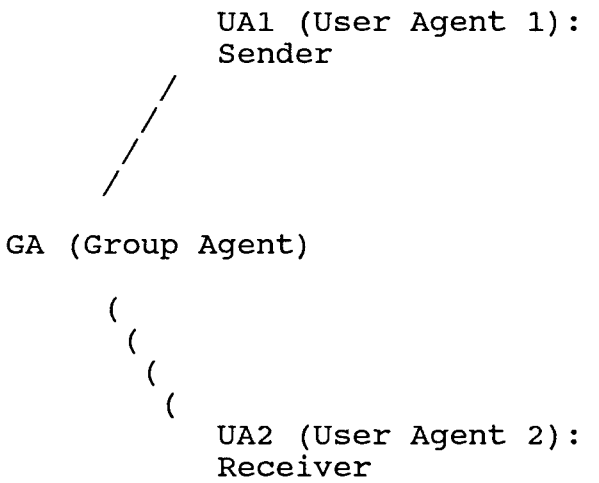
This section shall explain, with the use of an example, the distributed nature of EIES2 Smalltalk. This simplified example not only demonstrates the suitability of object model for a distributed CSCW system, but also shows the ease with which an application can be programmed without having to know where (i.e., on which agent or machine) a certain object is. Due to the space limitations we have selected a

simple application called "tell."

"Tell" of EIES2 performs a function similar to "write" on Unix, for immediate communication between group members. On EIES2, if it is desired to send a short message to another user, the "tell" feature can be invoked either through the menu or the command interface. The sender, has to provide the identification of the intended target user (i.e., receiver) and the text of the message.

As a result of the above, if the receiver is currently signed-on to EIES2, a window pops up on his/her screen. This window displays the text of the message and the identification of the sender. It also prompts the receiver to "reply" to the message or just continue doing his/her work, in which case the window disappears. On the other hand if the intended receiver is not on-line the text of "tell" is delivered to the user as a regular mail item. This mail item can be viewed later by the receiver.

With respect to EIES2 this example feature can be envisioned as follows:



The user agents (UAs 1 & 2 i.e., sender and receiver) or the GA (Group Agent) could all be on the same machine or different machines distributed across the network. Suppose you (the sender) want to do a "tell" to Joe (the receiver)

with the text this utility, the following Smalltalk message is sent to the class called "UserFace" (an instance of UserFace is associated with each UA).

```
myUserFace tell: 'Hi Joe' note: joe.  ..... (I)
```

The Smalltalk method invoked by this message (I) at UA1 is shown as under.

```
1.   class name UserFace
2.
3.   methods instance 'tell'
4.
5.       tell: aText note: aUser
6.
7.       eiesObject tell: aText from: user to: aUser
```

It can be seen, in the method above, that the receiver of the message (I) is an instance of the "UserFace" class, which represents the user agent (UA1) of the sender. With sender, receiver, and the text of the "tell" as arguments, this method sends a message to the object named "eiesObject" (see line number 7 in the method above). The "eiesObject" is an instance of the class named "EIES" and resides on the group agent (GA). The class EIES in a subclass of the class View. An EIES object provides a global system view to a particular group communication data base. An EIES object is assiciate with a particular agent, but it may reference Views from remote agents. A user agent logs into a particular group agent by constructing a remote reference to the EIES class on the desired agent. It obtains a valid remote reference to the EIES object upon successful login. The user agent can then access the "group memory" as its assigned roles permit. It should be noted that there is nothing in the method which requires the programmer to

provide any network related details. Since the object named "eiesObject" is on the group agent and not on the user agent, the CP finds this object, across network, and sends it the message.

The respective method on group agent is shown as under.

```
 1.  class name EIES
 2.
 3.  methods instance 'telling'
 4.
 5.  tell: aText from: fromUser to: toUser
 6.  | receiverUF |
 7.
 8.      receiverUF <- self userFaceForUser: toUser.
 9.      ( receiverUF isNil )    "User not on-line"
10.     ifTrue:
11.     [
12.        ( toUser getMessageView )
13.             itemSubmit: nil
14.             from:       fromUser
15.             newKeys:    ''
16.             newAssoc:   ''
17.             newTitle:   'undelivered tell'
18.             toUsers:    toUser
19.             withAttach: nil
20.             withText:   aText
21.     ]
22.     ifFalse:
23.     [
24.        receiverUF
25.             tellFrom: fromUser
26.             note:     aText
27.             replyTo:  fromUser.
28.     ].
```

This method validates the receiver and tries to get access to the its "UserFace" object (line 8). It checks if the receiver is currently signed-on (line 9). If the receiver is not on-line then the text of "tell" is submitted to the

receiver's "View" as a mail item (lines 10 - 21). If the receiver is signed-on at the time of "tell", the receiver's "UserFace" will not be Nil. In this case a message is sent to the "UserFace" object of the receiver, to inform him/her of a "tell" (lines 24 - 27).

Once again, the "UserFace" object associated with the receiver, more precisely the instance of the "UserFace" class for receiver (i.e., UA2) could be on the same machine as GA, or UA1, or another machine on the network. The location of UA2 is completely transparent to the application programmer. The receiver's "UserFace" is located by CP and the message shown in the above method, lines 24 - 27, is sent to it. The method invoked, as a result of this message is shown as below.

```
 1.   class name UserFace
 2.
 3.   methods instance 'tell'
 4.
 5.   tellFrom: aUserName note: aText replyTo: fromUser
 6.   | tmpWind reply |
 7.
 8.       tmpWind <- Window
 9.                      create: 3
10.                      xPos: 50
11.                      yPos: 5
12.                      xlen: 10
13.                      ylen: 10.
14.       tmpWind label:
15.           ( 'Tell message from:' concat: aUserName ).
16.       tmpWind put: aText.
17.
18.       wChoice clear.
19.       reply <- Wchoice input:
20.           'Enter reply or press return to continue >'.
21.
22.       ( reply sameAs: '' )
23.       ifFalse:
24.       [
25.           self tell: reply note: fromUser ].
26.       ].
```

The functions performed by this method are:

(a)- make a window on the receiver's screen (lines 8 - 13),

(b)- label the window with appropriate message and the sender's identification (lines 14 - 15),

(c)- put the text of the "tell" in the body of the window (line 16),

(d)- clear the bottom portion (called wChoice) of the screen and prompt the user to either reply to the "tell" or just continue (lines 18 - 20), and

(e)- if the user provides a message (i.e., a non Nil string object), it is sent back as a reply (lines 22 - 26). This repeats the same procedure all over.

It can be seen from this method that it is extremely easy to alter the user interface of EIES2. For example the receiver of "tell" can very easily change the location or size of the window.

4. CURRENT EIES2 STATUS

The current EIES2 development environment has a wide variety of tools which is the foundation upon which advanced features and enhancements are being developed. Current capabilities of the application model include an easy-to-use learner interface that supports: mail; conferencing with branching to allow sub-conferences by topic and/or responses; notifications; directory; keyword searching; options to LIST, SCAN, and VIEW TEXT; context-sensitive help; and an answer-ahead feature to provide short cuts; the ability to design forms; a microcomputer user interface; personal keywords to enable the user to define the way information is organized; pop-up windows; horizontal and vertical menus. A suite of activities are provided to support the on-line virtual classroom and group decision support. The system supports multimedia attachments.

Advanced applications currently planned include: group and personal calendars, project coordination, library services, group graphics design, and group hypertext.

/*

4.1 MAIL

The EIES2 mail facility provides a general capability for communication among users. Mail is organized as 'mail items" and "replies". Users who find themselves in a discussion that does not interest them may drop the item and not receive any further replies. Associated items are maintained independently to support relations among items across sub-systems and so that reply levels can be limited.

4.2 CONFERENCES

Conferences are communication structures organized by topic. Conferences support roles such as moderators, participants and observers. Members may be added or deleted and a complete transcript of the conference is kept. Conference items, like mail items, are organized as main items and replies. In a conference, however, there may be more than one level of replies. This organizes the conference into sub-topics and sub-sub-topics. The conference moderator may set the number of reply levels in the conference so that the branching of replies-to-replies can be limited. A conference can also be used as a general purpose discussion space for any group activity where more specific structures are not available or needed.

4.3 FORMS

The ability to communicate with forms rather than simple text is extremely useful in the CSCW environment. A form represents an implementation of structured messages as described by Malone [6]. In the AMIGO Model of group communications the EIES2 Form structure represents a Message Type in the context of Activities defined for group communications . A form can also be "attached" to a message or to a conference comment. When the user views the incoming message or the comment, he also will be prompted by the system to fill out the form. When the user completes the form, the information carried by the form will be

returned to the sender or other activity agent. In the current implementation, returned forms are delivered to the originator, and may be viewed, modified, or downloaded to the users work station in data interchange format (DIF) for further analysis.

4.4 ATTACHMENTS

Attachments provide a mechanism for associating activities with mail or conference items. The types of attachments currently supported include texts, items, forms, documents, and binary files. Other types are planned for "virtual classroom" and other applications. Text attachments, for example, allow a short introduction of a long text item. This allows receivers to easily decide whether they wish to read the long item or not. Item attachments allow forwarding of items received to other users. Binary files attachments provide an convenient means for micro-computer users to exchange software, documents, spread sheets, and other information in the CSCW environment.

4.5 ACTIVITES

Activities are special attachments that in addition to being seen or unseen by the receiver, they can also be done or undone. The system keeps track of a users activities in addition to new items.

4.6 KEYS

Keywords may be associated with any items on EIES2. Authors are prompted for keywords when sending mail or entering conference comments. In addition, receivers of items may add "personal" keywords thus providing an easy means for users to categorize items. For example, a user may add the personal key "urgent" to items that call for immediate attention. The keyword "projectx" might be used to maintain a list of items relating to a particular activity.

4.7 FILTERS

A filter is a mechanism that allows special handling of items that may be received in the future. Filters collect items that meet a particular search criteria independent of the subsystem of EIES2 they belong to. To the user, a filter seems much like a conference. Items may be received or otherwise operated on either in the filter, or in the particular subsystem where they are entered in the same way. This allows items from any subsystems that match particular search criteria to be handled by the user easily. The user is alerted to the presents of the new items at sign on and may process the items readily through the filter.

4.8 NOTIFICATIONS

Notifications provide a mechanism to keep the user informed as to actions taken by the system or other users that effect the user. Examples of notifications are the confirmation sent to a sender of a mail when the mail has been received by its addressee(s), and a user being informed that he has been added to a conference. Confirmations that another user received private mail from you, or that you were added to a conference are examples of notifications. You may also send one line notifications to another user.

4.9 TALK

The "talk" facility on EIES2 provides for real time interaction between members using a multiple window interface. Crises management and certain other problem-solving activities, where a quick consensus is required, often work better with this type of interface.

4.10 Distributed functionality

*/

EIES2 supports three kinds of distributed conferences.

1. tight - no replication.

2. loose - partial replication.

3. foreign - complete replication, i.e. netnews.

We have the mechanisms is the eies2 distributed object database to implement a wide variety of conference distribution strategies. With both tight and loose distribution we have tried to minimize user awareness of the distributed nature of the system by maintaining unique item and user names across the systems whenever possible.

The user agent only stores information pertaining to the uses interface to the system. It only keeps copied of the list of items the user is currently accessing. The user list of conferences and number of items waiting is kept on the group agent where the normally connects. Conferences in the users list may be local to that group agent or they may be located on remote eies2 group agents. Remote conferences that are tightly coupled do not copy any information to the local group agent. The number of items waiting and the texts are accessed on demand from the remote group agent. Loosly coupled remote conferences will have remote view copies that keep the number of items waiting locally and remote item copies to keep copies of the texts. When you add a remote conference, you choose whether to make it tightly or loosely coupled.

Internet users can be added to conferences and eies2 users can receive conference items as internet mail. We can link an EIES2 conference to a netnews news group. It is important, for connectivity sake, that we also add the ability to address EIES2 conferences from a mail only interface. But much functionality is lost when the group work environment is connected to communication only systems.

5. Summary

The Electronic Information Exchange System II (EIES2) is a second generation computer-mediated communications system developed at The New Jersey Institute of Technology with support from the New Jersey Office of Telecommunications and

Information Services (OTIS), the Commission on Science and Technology, and partnerships with AT&T Information Systems, Computer Sciences Corporation, and Jutland Institute of Technology in Denmark. EIES2 has been under development since 1984. It has evolved to provide an easy to use learner interface plus advanced features to satisfy both the first time member and the most advanced members.

EIES2 provides a comprehensive environment for electronic mail and computerized conferencing. The self organizing properties of the communications database help members cope with large amounts of on-line information. Conferencing coupled with a hierarchical item response structure organize communications into discussions and sub-discussions. Keywords and associations link items across different discussions. Personal keywords help members organize their communications. Members may drop any discussion or sub-discussion that does not interest them to minimize information overload. The global search capability allows members to find items anywhere on the system. Members can specify search criteria in advance for items of particular interest to them, these "filters" catch items as they are entered in the system and alert members to new items that are of particular interest to them. The on-line "forms" facility supports semi-structured messages for on-line data collection, this supports the development of group data bases and helps structure on-line group activities such as the "virtual classroom", automated facilities to support the office, the project, the library, etc. EIES2 provides extensive support for micro-computer users and can be accessed by virtually any type of terminal or PC. A simple menu system with consistent screen layouts provides extensive on-line help for new users. Commands and answer-aheads provide direct access to the full EIES2 capabilities for the advanced user.

The distributed architecture and support of modern standards will allow EIES2 to support a virtually limitless member community. To date, EIES2 nodes have been installed at NJIT, the University of Medicine and Dentistry of New Jersey, and the Jutland Institute of Technology. EIES2 currently runs on equipment from Hewlett Packard, Digital Equipment Corporation, Sun Microsystems, and AT&T. Currently, EIES2 operates under the UNIX operating system networking over TCP/IP networks. EIES2 networking

facilities have also been demonstrated over ISO, UUCP, and simple asyncronous line networks. There are plans to support VMS, AIX, MS-DOS and other operating environments.

EIES2 represents a set of tools for continued research at NJIT into computer-mediated communications. EIES2 provides it's own distributed Smalltalk environment to support the development and study of future system for computer supported cooperative work. It provides an integrated Abstract Syntax Notation 1 (ASN.1) compiler to support the current and future application layer standards.

EIES2 was developed as a research tool to become a "Laboratory without Walls" which would reach outside of the university environment and into organizations. It is currently in use at NJIT for project coordination, information exchange, group decision support, and electronic class room applications. EIES2 has been used to teach classes since 1988. It is now available to other organizations both as a utility run at NJIT and as software for sale. The application developers kit is also available to outside organizations. For more information contact the Computerized Conferencing and Communications Center (CCCC) at NJIT, King Blvd., Newark, NJ 07102, telephone (201) 596-EIES.

Further partnerships are being sought to apply the technology in the development of distributed CSCW applications, multi-media and PC user agent support, native Smalltalk integration, the development of application layer standards support, and in interfacing the EIES2 application functions with lower layer services in various network environments.