BUILDING CONVERSATIONS USING MAILTRAYS

Tom Rodden and Ian Sommerville
Department of Computing
University of Lancaster
Bailrigg, Lancaster
LA1 4YR. UK

Abstract

Current project support environments provide little direct support for group working. This paper postulates an architecture for future project support environments and describes an interaction metaphor appropriate for such an environment. Techniques for exploiting this metaphor in order to provide support for cooperative working are described. Finally a means of describing cooperation patterns within the environment is discussed and an editor which supports the construction of these description is introduced.

1. Introduction

Over the last few years considerable effort has been directed toward the development of Integrated Project Support Environments (IPSE) (for example; ISTAR [Dowson 87] and ECLIPSE [Alderson 85]). Such environments support the entire software process through the use of an integrated set of development tools covering requirements analysis to implementation and subsequent maintenance.

A common structure for project support environments is that of a tool set layered around an object management system (e.g. CAIS[Oberndorf 88] , PCTE [PCTE 85]) which provides mechanisms for maintaining consistency among project components to a degree which was not possible in older, file based environments. Current IPSEs offer considerably greater sharing of project information between tools than their predecessors. This sharing tends to be prescriptive in the sense that the object management system (OMS) limits tool interaction in order to protect project artifacts. To enable tool integration to take place by the sharing of common data, the object management system forces communication between tools (and users) to take place via the central object store. Both tools and users are protected from each other by *firewalls* which prohibit direct tool cooperation.

IPSEs have emerged as a method of supporting the development of large-scale software systems. Such systems are typified by their size and complexity and may demand the application of often large teams of experts in their design and construction. However, as early as 1975 Brooks [Brooks 75] observed an effect he termed "the tower of Babel problem". The tower of Babel effect occurs when the management of the communication overhead generated by a number of people working together becomes so great that it prohibits effective interaction.

Brooks suggested that the problems of team working are sufficiently grave that an effective maximum size for programming teams could be as small as ten members. It is very unlikely that future software systems which will be developed using the technology

offered by IPSEs can be constructed in any realistic timescale by as few as ten developers. We thus require tool support for interaction so that the effective working size of a team can be increased.

The work described here is concerned with interaction support tools. This paper introduces an architecture which encourages interaction between tools. The architecture and the interaction metaphor it supports is introduced the following section. Section 3 examines the development of this interaction metaphor to provide a cooperative interface. The handling of the information in such a cooperative interface is divided into two discrete problems the handling of *incoming* communication and the handling of *outgoing* communication, these are discussed in sections 4 and 5 respectively.

2. An Alternative IPSE Architecture

Current IPSE technology is based on a layered model. Tools are layered around some database or OMS and users interact directly with those tools. The users must know explicitly how to initiate tools and are solely responsible for managing tool interaction. By contrast, consider an alternative model of an IPSE, which has neither individual tools nor a kernel in the accepted sense. The architecture [Sommerville 88] consists of a federation of co-operating, distributed agents (Figure 1) which embody sufficient contextual knowledge to be invoked on an opportunistic basis. Agents can be considered as autonomous, asynchronous, immutable and may be cloned on several nodes in a network.

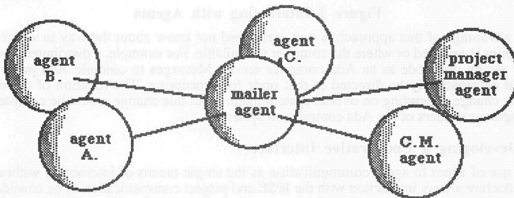

Figure 1 An Agent Based IPSE

An agent encapsulates both local data and the operations which may be performed by the agent and presents an external interface consisting of the set of allowed operations. Knowledge is held within this partitioned system knowledge base as objects [Rentsch 80, Wegner 87] Operations on data held by an agent are initiated by sending a message to the agent requesting some action. Controlled access of data allows sharing of information required by a diversity of agents. Because of the knowledge embodied in agents and the associated data encapsulation, users interact at a more abstract level and need not be concerned with how agents interact to solve a problem.

The view of a system as a set of autonomous, interacting, intelligent agents is similar to that perceived by users of electronic mail systems [Hiltz 81]. Given that there are increasing numbers of people who regularly use electronic mail, an appropriate interaction metaphor is that of electronic mail. Such a metaphor becomes even more pertinent because studies have shown [Losey 85] that workers spent approximately 60% of their time in communication-based activities.

By adopting the electronic mail model, agents may have a consistent interaction interface where exactly the same methods are used to send mail to system users, to file mail received and to initiate system actions. The conceptual view utilised by the Mailer is the filling in and 'sending' of forms to other agents. An agent initiates any action within the environment by passing a message object to another agent .

For example, to compile a program an agent (which need not be human) fills in a COMPILE form (object) this form is then submitted to the Mailer agent which *'mails'* the completed COMPILE form to the appropriate compilation agent (Figure 2).

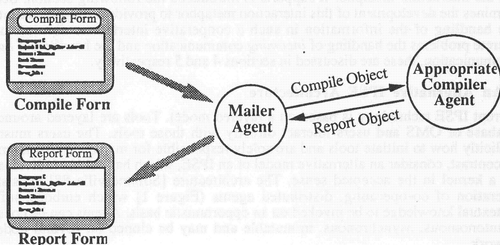

Figure 2 Interacting with Agents

The advantage of this approach is that users need not know about the way in which the compiler is initiated or where the compiler is available. For example, a distributed system may assign one node as an Ada compiler server. Messages to compile Ada programs could automatically be directed to that node for processing. The location of the node might change depending on overall system loading but this change would be completely transparent to users of the Ada compilation system.

3. Developing a Cooperative Interface

The use of agent to agent communication as the single means of interaction within the architecture allows interaction with the IPSE and project communication to be considered in a unified manner. This means that the provision of support for information transfer can become the central mechanism in supporting cooperative project development.

An underlying requirement within cooperative working support systems is the need for some structuring facility upon which to construct information handling systems. One approach which has proven useful as a method of structuring problem domains and implementing solutions is the Object Oriented approach. The principles of such an approach can be utilised as a structuring mechanism in the communication system for a IPSE. Objects are used to represent messages between users. Different types of message between user are represented by different message object classes.

These objects are presented to the user using an associated object browser interface [Sawyer 88] so that message objects are presented to the user as forms with attributes displayed as form fields and methods displayed as buttons. Message objects are organised in an inheritance network which the system exploits to provide a specialisation mechanism with various message forms inheriting methods and attributes from other forms.

The class definitions for message objects and the definition of the inheritance structure in which they reside are held in an object library used by agents to maintain a consistency of definition throughout the agents. The use of object classes to represent messages allows users to add structure to the messages transferred between them and subsequently allows the automated interpretation of these messages.

A natural division when discussing information transfer is to consider the control of the information involved in two distinct but related ways; the handling of *incoming*

information and corresponding support of *outgoing information.* A characteristic of incoming communication in traditional information transfer systems (for example, electronic mail) is the plethora of messages which must be processed by users [Malone 87]. This problem is amplified in an architecture where messages may be generated automatically[Hewitt 77]. A consequence is that incoming information must be structured so that the cost of information handling is minimised.

As with incoming information an equivalent set of problems is associated with the control of outgoing information.The most fundamental problem associated with outgoing communication is that of distribution. Distribution is the correct delivery of information to those users to whom it is of most interest and relevance. Any system aimed at supporting group interaction must address this problem.

4. Handling Incoming Information

Considerable research within the message handling community has been directed toward the development of efficient and reliable techniques for the transfer of messages. However, relatively little work has been done on interaction mechanisms for Message Systems [Malone 87, Hutchison 86]. Message Handling Systems currently tackle many of the problems generated by communication within environments by providing an efficient and reliable communication medium. Due to the historical emphasis on the transfer of messages, and the subsequent lack of development of user interface techniques, the presentation of information to the user tends to be both rudimentary and unstructured, generating an effect which has been termed *Information Overload* [Hiltz 85].

When information overload occurs, the flow of information is so rapid it makes it difficult for users to utilise the information relevant to their needs. To be effective, systems must give message recipients the ability to discriminate between those messages they wish to read and those of little relevance to them. Malone [Brobst 86] conducted several studies of how various kinds of information are shared in organisations. He outlines a number of approaches people use when reducing the information they need to process.

The most interesting of the filter approaches he describes is cognitive filtering, where the decisions are based on the topic of the message, and social filtering, where decisions are based on who supplied the information. Additional studies [Sumner 86] have shown that the majority of messages within electronic message systems are organizational, and that a significant amount of these are routine in nature.

Support for incoming information needs to provide facilities for message management, allowing the user to handling electronic messages as he would paper memos. The user, in addition to being able to compose and send new messages, should be free to file, reply to, or forward messagesand should interact via an amenable message interface which addresses the problems posed by information overload. Denning [Denning 82] suggests, that users of electronic communication systems feel most comfortable with the principles applied in current paper systems and that these should be adapted and augmented for computer based communication. This is the approach adopted in order to provide a familiar interaction metaphor for multi-agent systems.

4.1 Structuring the Incoming Information -The Mailtray Metaphor

Our structuring technique is based upon the notion of *Mailtrays* [Rodden 88] which messages flow into and out of as necessary. If we consider a paper based office which receives forms, memos etc.. for processing, it is standard practice for those items with some commonality to be grouped together and handled in a similar manner. A common grouping construct is the tray, so all mail which is normally processed in a similar way is placed in a particular tray, for example, all memos related to expenses, expense claim forms and budgets are placed in the expenses tray to be processed as expenses. Manual tray systems have a number of weaknesses, which the use of an electronic rather than a paper medium resolve. For example, in manual systems messages are often classified in

an arbitrary manner, since they can equally well be placed in any one of a number of trays. In the same electronic system however, a message may be placed in any number of trays and links between these instances automatically managed.

This notion of collating messages which are to be handled in a similar manner is the basis of our mailtray method. A user may define a tray as containing a particular set of mail items. When a message form arrives for a user the system places that message in the appropriate tray (or trays).

Each mailtray has a title, a number of attributes, and an action list which describes how forms should be processed. Additionally, each mailtray has an associated guard list controlling the nature of the forms held in the mailtray (Figure 3).

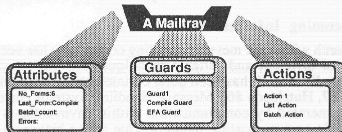

Figure 3: The Mailtray Construct

The decision whether or not a message is placed in a mailtray is controlled by a tray's *guardlist*. The guardlist is a list of predicates which are applied to the attributes of a message. If all the tests on a message's attributes succeed the message passes that guard and is appended to a tray. Any number of guards may be associated with a mailtray and a message is accepted if any of these guards are true.

For example a user could define an object which could be used for compilation. Such an object could have for example the identity of the source (whether it be a filename or and object id), the version of the compiler and the name of the project to which the compilation belongs. If a user wished to collect compilation forms from a particular project, say EFA_Flight_Simulator, then he could set up the attribute list to allow only project forms to be added to a tray as follows:

Guard *efa_comp* **class** *compile*
[
project = "EFA_Flight_Simulator"
];

For a form to successfully negotiate this guard it must be of class *compile* and have an attribute called project with a value "EFA_Flight_Simulator". A group of users can likewise define forms to meet their particular communication and interaction requirements. The tray system allows users the flexibility to collate these message objects irrespective of their class.

When a message is placed in a mailtray, the Action List for that tray is interpreted. Action Lists consist of a list of commands to be executed. Each command is of the form:-

Action Head -> [Action Body];

The action head consists of the action name followed by the class of messages to which the action applies. If the message is of the class appearing in the action head then the succeeding action body is executed. Each action body is written in a notation consisting of IF THEN and assignment statements in conjunction with message handling primitives (mail, save ,forward etc.). The notation used for describing actions is aimed at extracting simple decision making information from the messages attributes while more complex and specific procedures are handled by methods associated with the messages. For

example, if a user wished to read mail of class *progress* only when he had ten messages of this form his action list for the appropriate tray might contain the action *batch* :-

```
batch : progress -> [
                        messages = messages +1;
                        IF messages = 10
                           THEN
                              [
                               Flag;
                               messages = 0;
                              ]
                        ];
```

Messages is a user defined attribute associated with the tray to which this action belongs. **Flag** is a message handling primitive which alters the mailtray icon to inform the user the tray requires attention. In addition to Flag, which alters the way trays are displayed to the user a second output primitive **write** allows information to be written to a standard output channel. Other primitives include **Forward** which allows messages to be forwarded to a number of users, **File** which stores messages in a file, and **Reply** which sends a reply to a message sender.

5. Handling Outgoing Communication

A number of researchers have examined the problems associated with the distribution of information and have postulated possible strategies for information distribution (or sharing). Research within the context of CSCW can best be described as representing a variation in rigour ranging from the very structured approach of Winograd and Flores who utilised speech act theory in their Coordinator system [Winograd 87] through the semi-structured approach of Malone's Information Lens [Malone 86] and Conklin's gIBIS system [Conklin 87] to the freely structured world of current Hypertext systems such as Notecards [Trigg 86]. The variation in rigour so predominant within CSCW is indicative of the need to express a variety of cooperative tasks. Indeed, both the success and most significant drawback of Winograd's Coordinator system is its strict control of rigorous communication strategies.

Two distribution techniques are provided by the Mailer system, namely register distribution and conversation based distribution . The first of these is the use of a register to support the contextual delivery of information based on the values associated with the attributes of a particular message. This register allows the definition of abstract destinations which are independent of users, these abstract destination are synonymous to roles within an organisation. A graphical interface is provided to allow the definition of both these abstract destinations and form/destination relations(figure 4).

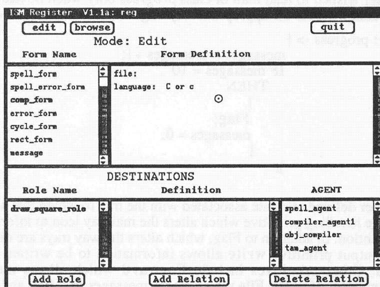

Figure 4 A graphical register interface

The register holds details of the current assignment of abstract destinations (roles) to actual agents. These roles can be moved between agents by altering their assignments within the register. The register approach outlined above allows the support of the communication patterns needed by loosely coupled work groups who cooperate to solve incompletely defined problems

Group activity is also supported within the environment in a more direct manner; particularly for those closely coupled groups following a known documented procedure for cooperation. This form of cooperation has traditionally been addressed by systems developed from Searle's linguistic speech act theory[Searle 75] and follows a model based upon person to person interaction. Our work utilises the object model with its computational history to describe and control information distribution.

5.1 Defining Conversations Using Trays

Consider the situation where a group is cooperating on the completion of a task whose solution path is known (such as voting). Such tasks often have existing documented procedures for thier solution. In these cases it is possible to specify in detail what is expected of each of the participants in a cooperative task.

This class of problems represent a considerable proportion of recent CSCW research. Most of the systems developed to tackle this problem domain have evolved from research in linguistics and are strongly influenced by Searle's work on speech act theory within discourse analysis. Notable systems utilising or influenced by speech act theory include the Coordinator Project [Winograd 87], the Chaos Project[De Cindio 86], and the COSMOS [Wilbur 88] and AMIGO [Danielson 86] projects.

As in systems based on speech act theory our final distribution mode makes great use of typed message interchange as a means of cooperation. Systems based on speech act theory generally attempt to provide a central script which describes a cooperative conversation. This script is often interpreted centrally to drive the conversation described. On the other hand conversations described within our model concentrate on capturing the *communication paths* between agents and decentralises the description of the actions performed by the roles.

Our overall architecture (and view of cooperating group members) is that of a federation of intelligent agents where an agent may be a process or a group member. This view of cooperating individuals as intelligent agents passing messages to solve a common task is analogous to the object based computational model. Consider an object which make use

of the services provided by other objects . The requesting object sends a message to the object whose service it requires. The object has an method associated with message and this method is interpretted with any results are sent back to the requesting object (figure 5)

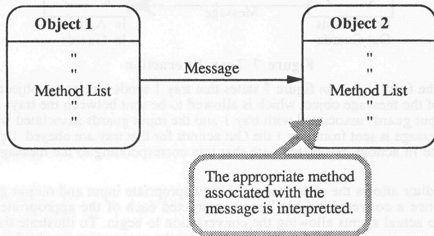

Figure 5 Object Interaction

In the same manner an object oriented program consists of a set of objects which are linked by message passing For example a program might be designed to monitor various information within a car and display tyre pressure, speed, fuel level and various engine information (e.g, revs/min). Such a program could be realised by the collection of objects shown in figure 6.

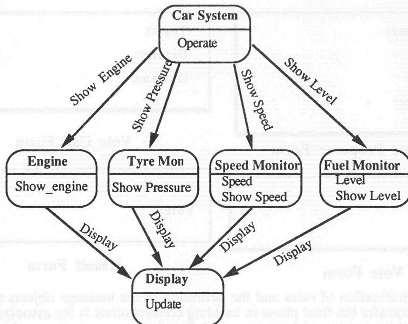

Figure 6 An object Oriented Program

This observation that the object oriented computational model relies upon message passing allows the model to be exploited to describe entities which cooperate by message passing. Thus, cooperative activities can be described as a collection of message passing agents. Such a collection of agents is termed a *conversation.* The tray metaphor is extended to allow the necessary detachment between actual agents and the roles agents play within conversations. Conversations are constructed by joining trays together and

are analagous to object oriented programs The interchange of a message between two trays within a conversation is shown in figure 7:-

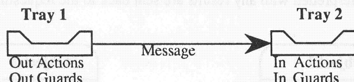

Figure 7 Tray Interaction

A link of the form shown in figure 7 states that tray 1 sends a message object to tray 2. The type of the message object which is allowed to be sent between the trays is defined by the output guards associated with tray 1 and the input guards associated with tray 2. When a message is sent from tray 1 the Out actions for that tray are obeyed. Upon arrival at tray 2 the In actions associated with that tray corresponding to the message type are obeyed.

The tray editor allows the construction of the appropriate input and output guards and actions. Once a conversation has been constructed each of the appropriate trays are assigned to actual agents allowing the conversation to begin. To illustrate this process consider how voting is supported. We can consider the cooperation involved as follows:-

Appropriate roles are:-

- Chairman
- Teller
- Voter
- Proposer

Each of these would have an associated tray, appropriate forms could be:-

Project name:
Date:
Topic:
Vote Raiser:

Vote Proposer Form

Topic:
Details:
Options:

Vote Call Form

Topic:
Vote:

Vote Form

Topic:
Vote:

Result Form

After the identification of roles and the definition of the message objects required for information transfer the final phase in building conversations is the association of roles with trays and the linking of these trays to define the communication paths. A possible strategy for linking trays is shown in figure 8:-

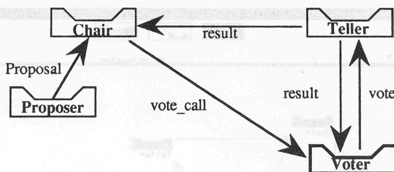

Figure 8 A voting Conversation

Actions can be associated with each Mailtray. However the use of conversational distribution highlights the need for an addition to the Mailtray structuring technique. If two trays are linked together it is likely that a user may wish certain actions to take place upon sending of a message, for example, he may wish to keep a copy of any message sent and its destination. To achieve this the concept of output actions are added to Mailtrays. Output actions are exactly the same as the actions discussed in section 4.1 except that they are interpreted whenever a message is sent via a tray.

A typical incoming action for the vote collators tray could be:-

```
count_results : result -> [
                      IF vote = yes THEN
                         [
                         yes_vote_count := yes_vote_count +1;
                         ]
                      IF vote = no THEN
                         [
                         no_vote_count := no_vote_count +1;
                         ]
                      ];
```

Tray conversations are stored in a *conversation agent* one of the autonomous agents within the IPSE architecture described in section 2. The conversation agent holds the tray information and details of the links joining them. A user can request a tray which is part of a conversation and the tray is sent to him as a message object. which the mailtray interface interprets. Any information sent via a tray which is part of a conversation is first sent to the conversation agent which then decides the ultimate destination of the information from the conversation details stored within its local knowledge base.

Tray conversations are created by directly interacting with the conversation agent. using a conversation editor. The editor supports the graphical construction of Mailtray conversations. The user may define a number of trays and the link these trays together and define the type of message which flows between them. The editor consists of two panels the control panel which permits the addition of new trays and links and the conversation panel which allows tray conversations to be drawn (Figure 9):-

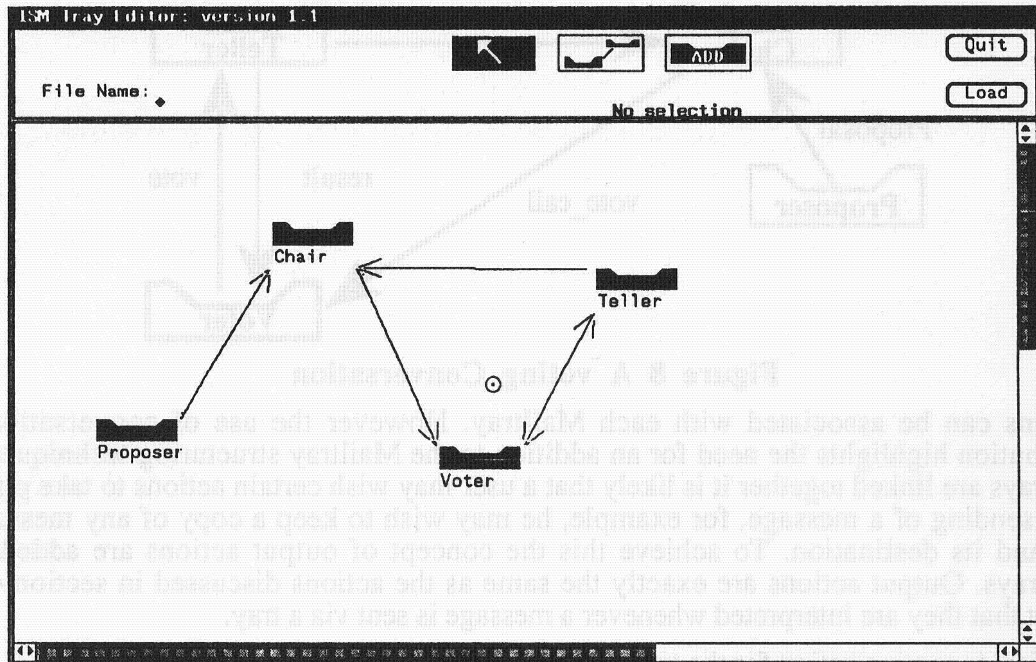

Figure 9 The Conversation Editor

Each of the tray icons has an associated menu which allows the definition of guards and actions associated with the corresponding messages. Each of the arcs have similar pop-up menus which allow the addition and deletion of the message types which link trays

Trays defined using the tray editor and forming part of a tray conversation are assigned to actual users in order to realise a tray conversation. For example the teller tray defined in the voting conversation shown in figure 7 would be assigned to a particular user and become part of the users mailtray interface.

6. The Mailtray System

In the previous sections we have highlighted how the Mailer system handles both incoming and outgoing communication in such a manner that cooperation is encouraged. The developed Mailtray system consists of a number of distinct components. Each component is implemented as a separate autonomous agents within an agent based environment similar to the one in figure 1. The agents within the mailtray system are:

- **A Mailtray Interface Agent**
  The Mailtray interface is used to construct a tailored organisational interface which automatically collates messages into appropriate groupings. Actions can be associated with each of the trays allowing the semi automatic handling of certain classes of message.

- **A Form Register Agent**
  The form register agent allows forms of a particular classes to be registered for automatic distribution to either agents or roles which can be transferred between (assigned to) agents

- **A Conversation Agent**
  Trays may be linked together to form tray conversations. These tray conversations define group activities in terms of the trays used by the tray interface agent. The conversation agent allows these conversation to be constructed by the use of a tray editor Trays which are combined by the tray editor may be assigned to particular users and then become part

of that users tray interface agent. The conversation agent maintains the details of these tray conversations and provides the necessary routing for any relevant information.

The above group of agents are organised as shown in figure 10:

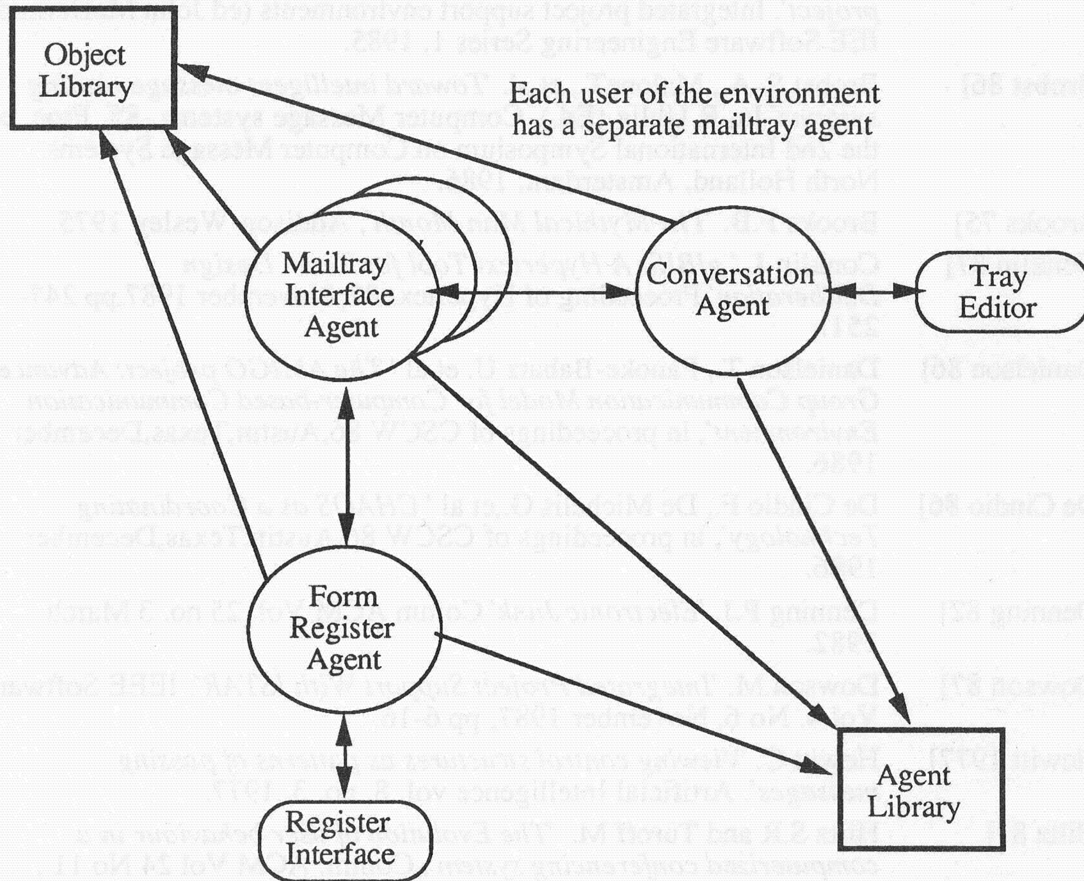

Figure 10: The Mailtrays System Architecture

The object library and agent library are central depositories which contain information on the object classes and their class hierarchy and agents and their locations in the system. This information is available to all agents including those shown since both the message object class definitions and agent information are often required to be consistent throughout the environment.

7. Summary

Modern integrated project support environments are deficient in their support of groups or teams cooperating to tackle a common task. The work presented in this paper is aimed at redressing this imbalance by providing support for communication as an intrinsic part of the interaction metaphor for future IPSEs.

Communication support is provided for both incoming and outgoing information. Support for incoming information is provided by the use of a structuring technique aimed at minimising information overload, a classical problem within the message handling community. The structuring technique supported is based on the notion of Mailtrays which automatically collate incoming information to match the user needs.

Support for outgoing information is provided by the development of two distinct distribution techniques. The first of these uses a register in order to associate users who wish to work together without a predefined work pattern. The second distribution technique allows conversation to be constructed by joining the trays in different users

mailtray interfaces. The complete system is realised as a collection of autonomous agents which coordinate their activities by message passing.

8. References

[Alderson 85] Alderson A., Bott M.F., Falla,M.E. *'An overview of the ECLIPSE project'*. Integrated project support environments (ed John McDermid), IEE Software Engineering Series 1, 1985.

[Brobst 86] Brobst S. A., MaloneT., et al. *'Toward intelligent message routing systems'* In, R Uhlig (Ed.), Computer Message systems -85. Proc. of the 2nd International Symposium on Computer Message Systems. North Holland, Amsterdam, 1986.

[Brooks 75] Brooks F.B. *'The Mythical Man Month'*, Addison-Wesley 1975.

[Conklin 87] Conklin J. *' gIBIS: A Hypertext Tool for Team Design Deliberation'*,Proceeding of Hypertext 87, November 1987,pp 247-251.

[Danielson 86] Danielson T., Panoke-Babatz U. et al *' The AMIGO project: Advanced Group Communication Model for Computer-based Communication Environment'*, in proceedings of CSCW 86,Austin,Texas,December 1986.

[De Cindio 86] De Cindio F., De Michelis G.,et al *' CHAOS as a Coordinating Technology'*, in proceedings of CSCW 86,Austin,Texas,December 1986.

[Denning 82] Denning P.J. *'Electronic Junk'* Comm ACM Vol. 25 no. 3 March 1982.

[Dowson 87] Dowson M. *'Integrated Project Support With ISTAR'*. IEEE Software, Vol 4. No 6, November 1987, pp 6-16.

[Hewitt 1977] Hewitt C. *'Viewing control structures as patterns of passing messages'*. Artificial Intelligence vol. 8, no. 3, 1977.

[Hiltz 81] Hiltz S.R and Turoff M. *'The Evolution of user behaviour in a computerized conferencing system'*, Comm. ACM Vol 24 No 11 , November 1981, pp 739-752.

[Hiltz 85] Hiltz S.R , Turoff M. *'Structuring computer-mediated communication systems to avoid information overload'*, Comm ACM Vol 28 No 7 July 1985.

[Hutchison 86] Hutchison D., Armitage R., Muir S.J.*' A User Agent for the Unix Mail system'*., Data Processing Vol 28 No 10 Dec 1986.

[Losey 85] Losey C *'Electronic Message Systems For More Effective Management '* IEEE Trans on Professional Communication,Vol. PC-28 No. 3 Sept 1985.

[Malone 86] Malone T.W., Grant K.R., Turbak F.A. *'The Information Lens: An Intelligent System for Information Sharing in Organisations'*. ACM CHI'86 Proceedings, 1986.

[Malone 87] Malone T.W., Grant K.R., Turbak F.A., Brobst S.A., Cohen M.D. *'Intelligent Information Sharing Systems'*. Comm. ACM vol. 30, no. 5, 1987.

[Oberndorf 88] Oberndorf P.A. *'The Comon Ada Programming Support Environment (APSE) Interfaces Set (CAIS) '*, IEEE transactions on software engineering, Vol. 14 No. 6, June 1988, pp 742-748.

[PCTE 85] PCTE Project team *'Overview of PCTE: A basis for a portable Common Tool Environment'* Esprit, 1985.

[Rentsch 80] Rentsch T. *'Object Oriented Programming'*. SIGPLAN Notices, OOPS 80, 1980.

[Rodden 88] Rodden T., Sommerville I. *'Mailtrays: An Object Orientated Approach to Message Handling '*, Presented at EURINFO 88, First European Conference on Information Technology for Organisational Systems, Athens 16-20 May 1988.

[Sawyer 88] Sawyer, P., and Sommerville, I. *'Direct manipulation of an object store'*, Software Engineering Journal, 1988, 3, (6), pp214-222.

[Searle 75] Searle J.R. *'A Taxonomy of illocutionary acts'*, in K.Gunderson (Ed.), Language,Mind and Knowledge, Minneapolis, University of Minnesota, 1975.

[Sommerville 88] Rodden T., Sawyer P., Sommerville I. *' Interacting with an active, integrated environment'*, Presented at ACM SIGSOFT symposiom on partical software development environments, Cambridge, MA., Nov 1988.

[Sumner 86] Sumner M.*'A Workstation Case Study'*, Datamation, Feb 15 1986, pp71 - 79

[Trigg 86] Trigg R., Suchman L., Halasz F.*'Supporting Collaboration in Notecards'*, in proceedings of CSCW86, Austin, Texas, December 1986.

[Wegner 87] Wegner P. *'Dimensions of Object-Based Language Design'*. OOPSLA'87 conference proceedings, ACM press, 1987.

[Wilbur 88] Wilbur S.B., Young R.E. *'The COSMOS Project A Multi-Disciplinary Approach to Design of Computer Supported Group Working'*, in R. Speth(ed) EUTECO 88: Research into Networks and Distributed Applications, Vienna, Austria, April 20-22,1988.

[Winograd 87] Winnograd T. *'A language/action perspective on the design of cooperative work'*, Stanford University Department of Computer Science Technical Report , STAN-CS-87-1158.