MIT/LCS/TM-210

SOFTWARE FOR THE "ROLES" PEOPLE PLAY

Irene Greif
February 1983

MIT/LCS/TM-210

SOFTWARE FOR THE 'ROLES' PEOPLE PLAY

Irene Greif

February 1983

Software for the 'Roles' People Play

Irene Greif
January 14, 1983

Abstract

Office work consists largely of cooperative efforts by numbers of people. To support such work, applications programs can be designed as "multi-person" systems organized around notions of "roles" and "working relationships." A group of co-workers can then describe to the system their agreed upon roles in a project as well as the working relationships among those roles. Based on this description, application software can provide support for communications protocols and access control that is tailored to the working situation. As working relationships evolve, these descriptions can be modified so that the software will continue to meet the needs of the users.

The paper presents an approach to office systems research emphasizing the development of software modules that can be used to build end-user application programs. The requirements that "multi-person" applications place on this software architecture are discussed in the context of a series of examples of multi-person activities, including joint document writing and calendar management.

KEYWORDS: office automation; desk-to-desk conferencing

This paper will appear in the Proceedings of INTERFACE '83
Miami Beach, Florida, March 21 - 24, 1983

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract Number N00014-75--C-0661.

Table of Contents

1. Introduction

Commercial products combining telephone and computer terminal, or providing broadband networks for voice and data communication, pave the way for increased integration of communication and computing. But the power of this communication medium will not be realized simply by networking together computer terminals on every desk. There will have to be an accompanying change in end-user software to support cooperation and communication. Currently, "integrated office systems" products provide any number of support tools for *individuals* -- word processing, data retrieval and personal file management, spread sheet calculations, graphics design and calendar management, for example -- but only one facility, electronic mail, for *communication*. Future communication facilities will be integrated more uniformly throughout all applications software to provide support for such *multi-person* activities as

- co-authorship of documents,
- group budget planning,
- design of complex artifacts,
- real-time monitoring of manufacturing facilities,
- traffic control

Such facilities cannot be built from existing software for individuals but require redesign of basic system components to allow their use in both single and multi-person settings.

This paper sets forth an approach to office workstation software architecture and explains how that architecture is influenced by the requirement that there be end-user facilities for cooperative activities. As the title of the paper suggests, one of the major concerns is determining how to build software that can support an individual in the different "roles" he plays: he may be a manager on one project, a group member in another; he may be a co-author of a document; he can chair a meeting, or simply participate as an observer. In each case his actions may be different, and his working relationship with others may be changed. The support available to him on his workstation must change accordingly.

2. Software Architecture for Office Workstations

The Office Automation Group at M.I.T.'s Laboratory for Computer Science has formulated and been guided by a philosophy of "functional office automation," aimed at identifying business goals of organizations that can be supported by computer-based office systems. This view has led to

development of new office analysis techniques [Sirbu et al, 1982; Kunin, 1982] as well as a system approach to design of office workstations.

Our office studies have shown that each office and each individual will place unique demands on an office workstation: there can be no single software package that will satisfy all of these demands. New software will continue to be designed and built, and existing software will have to evolve as working environments change. Thus our task in office workstation research is not to design and build office software, but rather to understand the fundamental principles of office system design. In our view, the office workstation architecture is not a set of integrated end-user applications, but rather a collection of more basic software modules that support the programmer in building end-user facilities. The importance of of this approach is twofold. First, it eases the task of programming each time a new application is required. Perhaps more importantly, it encourages consistent styles of programming across applications. If programmers use the same basic tools, their application programs are more likely to constitute an integrated environment with consistent command language, interface and data representation. This assures that end-users can carry over skills from one application to another: they know how to ask for help, where error messages can be viewed on the screen, how to terminate a session.

Our ETUDE and ECOLE projects [Hammer, 1981] illustrate these design principles. The ECOLE integrated office workstation consists of a number of software packages that determine the appearance of the interface of new application programs. The design of these packages was influenced by the requirements for ETUDE, the text processing subsystem. ETUDE is a document editer designed to be easy-to-learn as well as easy-to-use. It is difficult to meet both of these criteria in one interface: menu selection and extensive help messages for naive users can get in the way of experienced user; succinct abbreviations preferred by experts are incomprehensible to the new user. A review of the "folklore" of user interface design [Good, 1981] led to a design which meets both requirements. ETUDE has a command language of English-like phrases (in verb-modifier-object format), reserved keys for "help," "undo," and "cancel," and menus of commands. The menus and help are optional -- when called up, menus appear in a standard location and help messages are placed on the screen without obscuring the user's cursor. Command completion by the system allows short forms of commands for the experienced users.

To support ETUDE, we instalied two interface packages on ECOLE: a table driven command parser and a window management system. The command parsing routine operates on a table of commands indexed by verb. For each verb, the table contains help and parsing information about the rest of the command phrase, and pointers to executable code to be used once the command is

parsed. The command parsing routine communicates with the window manager when it recognizes that help or menus must be displayed. The window manager determines the location of new windows based on the kind of information it is sent and the current state of the screen.

Thus the command parser and window manager cooperate to provide screen organization and command language support. By changing the table of commands one can get similar support for other applications. Both programs were used for ETUDE and again in personal calendar system, PCAL [Greif, 1981]. Commands in each case have similar phrase structure: in ETUDE one can MOVE 5 LINES or DELETE NEXT PARAGRAPH; in PCAL one can LIST NEW APPOINTMENTS or SCHEDULE [A] MEETING.

Human factors tests [Good, 1982] show that ETUDE is indeed easy-to-learn, without forcing tedious menu selection on the experienced user. The ECOLE command parsing and window packages embody the principles of this design including the command language orientation (no pointing device was available), the phrase structure, and reserved keys for special functions. Using these packages, the same *kind* of interface can be installed on other applications so that text processing, calendar and spread sheet facilities all present the same kind of interface. The ETUDE/ECOLE work factors out the user interface decisions so that changes to the interface, experiments with alternative interfaces and addition of new equipment such as pointing devices can be accomplished easily and uniformly across applications.

ETUDE also supports two views of a document, its *logical structure* of e.g. paragraphs, sections and chapters, and its *outward appearance* of e.g. lines and pages. The outward appearance is maintained on the screen to show the user how the document will appear on paper. Since the user may need to refer to aspects of the logical structure as well, a summary of the logical structure is presented along side the document as shown in Figure 1. ETUDE must maintain the accuracy of both views after changes to either are made.

To maintain both views ETUDE requires data base support incorporated in ENCORE, the "object management" package of ECOLE. ENCORE is a cross between a file system, in which *objects* are stored, and a data base in which *information about objects* is stored. ETUDE documents cannot simply be stored as strings of characters in a file. A document is a structured object that has outward appearance attributes associated with its components. In general, ENCORE facilitates the storage and retrieval of small, non-uniform objects, providing services required for many office applications, but not adequately supported in either conventional file systems or data base management systems. Relationships between objects can be expressed, for example, the containment of paragraphs in sections or the placement of lines of text on a page; operations for maintaining multiple editable views of documents can be implemented.

To summarize, we study applications and build prototypes in order to understand the software infrastructure of the office workstation. Our main results are design guidelines and principles of software organization. More important than ETUDE's easy-to-use and easy-to-learn document editing features, is the fact that its implementation isolated features of the design in separate software modules. In the case of ETUDE these include object management and an interface that integrates command language interaction with optional menus.

Our current focus on cooperating office systems is similarly motivated. We have observed that cooperative work is an aspect of office system development that is poorly understood and inadequately supported. As we develop and test prototype multi-person application systems we are also identifying the basic system components that will make it possible to build other similar systems easily. The rest of this paper contains examples of end-user facilities for multi-person activities and discussion of the workstation packages that will be needed to support these facilities.

3. Multi-Person Activities

People have been working together for a long time without computers. For example, they convene at one location for a conventional meeting, they meet-at-a-distance via telephone conferences, they hold video conferences, or they meet-over-time through correspondence. On the computer-based office workstation, an electronic mail or message package is the current standard mechanism to support such group work. An extension of electronic mail known as "computer conferencing" provides additional features for organizing messages when a large number of people conduct a meeting over time.

As more office work and data is kept on-line, we see increasing opportunities and need for additional kinds of communication support. As normal desk equipment comes to include both telephone and computer-based workstation, desk-to-desk conferencing will become a powerful mode of communication: co-workers will share on-line data as simply as they converse by telephone. For many kinds of work at a distance the overhead of video connections and studio teleconferencing will be unnecessary -- people will have the convenience of working with others without leaving their desks or giving up access to their (on-line) files.

This kind of support is not now in place. For example, many people use word processing facilities for composing and editing their own documents. However, co-authors writing a document together on-line are left to their own devices to maintain the integrity of the document. If both edit it at once, there is a high probability that the changes of one will be lost. If one makes changes to be read and approved by the other it is difficult (if not impossible) for the reader to find out where the changes are

and what the old version looked like. The author who has made revisions typically will send a message to his co-author saying, first, where the new version can be found (in what "file") and then describing the changes. If two authors working together at a distance "link" their terminal screens in order to make the document visible to both simultaneously, again, they must devise and enforce their own protocols for taking turns writing.

In this section we suggest a number of ways in which co-workers can be supported in their cooperative work by extended facilities for coordination, control of sharing, meeting support and automatic message generation. The examples span two kinds of meetings: "meetings over time" in which co-workers act asynchronously on shared data, and "simultaneous" meetings in which co-workers use the computer terminal to provide a shared workspace for a meeting in real-time.

3.1. Communication and Coordination

In the joint document writing situation described above, the co-authors could be relieved of some of the protocol enforcement if their text editing systems knew about their intended *working relationship*.

For example, if the authors have agreed to notify each other of their changes, notification could be sent by the system to co-authors after each editing session stating that revisions have been made. Thus individuals can simply terminate editing sessions as they would when working on personal documents: all follow-up messages are sent automatically.

If two authors are editing at once several kinds of system intervention might be appropriate: all but one author may be prevented from editing; the authors may be able to work at once on different parts of the document; the authors might be notified of each other's presence on-line in case they want to link their terminals and work together. In this last case they would probably initiate a voice conference and use the computer as a shared workspace.

We mentioned above that linked terminals provide no coordination support. A proper meeting interface could provide a shared workspace in which to display the document on each user's terminal. Instead of one cursor for pointing at text, there could be two distinguishable cursors, one for each author. The meeting interface would prevent both authors from moving their cursors to the same location and writing in the same place.

If a large number of people were trying to work on the document at once multiple cursors might be inappropriate. Instead, one person might *chair* the meeting and other *meeting participants* could request the floor. Passing control to a meeting participant would allow him to move the cursor in the

shared space and make changes to the document.

Clearly these kinds of communication and coordination protocols could arise not only in joint document writing, but in joint design (with graphic display), calendar management, project management, on-line debugging, or teaching facilities. People working asynchronously would like to establish at the outset certain conventions for *notification*. People working simultaneously might choose one of a number of *meeting support* structures, such as the two described above -- the informal structure with minimal conflict protection or the more formal chaired meeting.

3.2. Change Histories

An editing author can also be relieved of the chore of describing his changes: the system could translate his editing commands into operations that store old and new versions of changed text. When his co-author reads the document the text processing system could let him move from one change to another and examine old and new versions of each part of the text. This is accomplished by having the text editer keep a *change history* recorded through multiple versions of document components, while the first author is editing the document. Many office applications require communication about *changes* to information; the change operation itself can be made to facilitate the communication through *automatic notifications* and *recording of change histories.* Some people might wish to be notified every time a new version of an object is added to the data base. Others would like to be informed of the option of seeing multiple versions when examining the object but don't need special notification. Most people would only wish to see the most recent version. These are all options that should be describable and implementable in multi-person systems.

The recording of change histories requires storing *multiple versions* of some objects in the data base. In a situation where changes are being reviewed, one might view any number of older versions as well as the most recent one. The presentation of the multiple versions is up to the particular application program: a joint document writing interface might display old and new versions side by side or alternatively old text could be crossed out with new text written in between the lines.

3.3. Privacy and Sharing

Some data should not be shared. In joint document writing, each author may want to write some *private* comments on a document not visible to his co-authors. Although the document and all associated comments may be stored together in a single data base, each author should be presented with his own view including only what he is entitled to see. During a simultaneous meeting, only text that is viewable by all participants should be shown in the shared space. The need for multiple views was identified in our earlier work with ETUDE documents [*Cf.* Section 2]. However, now the views

may also be determined according to additional information about *ownership* of data and *authorization*. This puts additional constraints on how the data base must be organized and on the kind of information that the data base must be capable of storing.

Occasionally in a simultaneous meeting with the co-authors, an author may want to see his private comments, without making them visible in the shared workspace. Meeting support interfaces should provide at least two workspaces, shared and private. The data presented in each space is processed to provide the appropriate "views." An example of this kind of meeting interface appears in our RTCAL [Greif, 1982] facility for sharing calendar information during a meeting-to-schedule-a-meeting. Participants provide filtered views of their calendars to a meeting support program so that a shared space can be displayed showing when the group as a whole is free, without revealing details of individuals' appointments. During discussion of possible meeting times, individuals can view full details of their calendars in the private workspace as needed for decision making. Figure 2 shows the screen during an RTCAL session.

3.4. Restrictions on Operations

Suppose John Doe is writing Section 1 and only commenting on Section 2 of a document. In Section 1, John's role is *author*, and he has full access to all editing commands. In Section 2, his role is *commentator,* and he can only add comments and suggest changes. The operations available to him reflect the working relationship that he has agreed to with his co-workers.

Restrictions on operations often arise in cooperative office work. In our calendar facilities there are a number of different restricted roles: any laboratory member can reserve a conference room time slot, but cancellation of other people's reservations is prohibited; professors may allow students to make appointments on their calendars during office hours, but not at other times. To enforce restrictions the system can refuse to complete a request, or alternatively, it can translate restricted operations into allowable transactions. An attempt to schedule an appointment might be translated into a request for an appointment; an attempt to rewrite a phrase in a document might be translated into a comment on the document suggesting a change. The data base capability for storing multiple versions of objects can be used to support this translation process by storing alternative versions instead of actually changing objects.

3.5. Organizing the Data base

Information in a data base is stored and retrieved according to constraints specified in a "data base schema." In ENCORE, each schema includes definition of the *structure of an object* and the allowable *operations on the object*. For a document, the structure part of the definition might specify that the document contains paragraphs and comments. The operations might include reordering paragraphs within a section, reading or deleting a chapter, and so on.

Multi-person activities require inclusion of additional information in the schema:

- ownership information may be associated with components of the object
- multiple versions may be kept for components of objects
- notification may be triggered by certain operations
- views can be composed by selecting and processing components of objects.

As working relationships in a group evolve, there may be changes in the amount and kind of information sharing that group members wish to establish. Implementing these changes will require modifying the data base schema. For example, an author of a document may decide after writing a first draft that he should be co-authoring the document with a colleague: he will have to incorporate information about their working relationship into the document data base. This ability to change the schema applies to all aspects of the data definition, including specification of which objects are stored with multiple versions, which operations trigger notification to co-workers, etc. Clearly, the extensible data base facility required for multi-person applications is beyond the state of the art in data base management systems and constitutes a research project in itself.

4. Roles and Working Relationships

The previous section examined needs for communication protocols, access rights control, multiple versions and change histories, and automatic notification. We suggested that the data base schema include information about the way data is stored, the amount of ownership and authorization information to collect with each item and the kinds of data base changes that might trigger communication.

A *role* is a selection from a set of options in each of these areas. The details of a role may be extremely complex, and require intimate knowledge of all of the capabilities of the data base definition language.

We believe that the end-user should be able to add and modify new roles, and assign new people to

roles. This aspect of workstation development is one of the primary targets of our research. It is an area of software development and end-user support that has not been given sufficient attention in other related work, including research on office systems and intelligent, so called "user-friendly," computing environments. In a well-designed multi-person application, there will be certain built-in roles and intuitively understandable working relationships about which the user can learn details incrementally as he goes. But the ability to modify these definitions is most important. In addition to building support for the system programmer, we must develop an end-user interface to the roles and working relationship package that will allow an office worker to control his own working relationships.

There is some precedent for this kind of end-user facility in the SCRIBE document formatting system [Reid, 1980]. Document designers write long complex definitions of document types such as *article, report* or *letter* specifying choices of font, spacing, allowable document components (e.g. chapters exist in reports but not in articles) and so on. End-users typically simply choose a fixed document type and work within it, never learning all the details of the definition. The document type names suggest correspondence to familiar document types so that users can anticipate the general appearance of a document formatted with SCRIBE. If a writer of a *report* decides he would prefer underlined chapter headings to the built-in bold face headings, he can *modify* the report definition with respect to chapter heading font choice without redefining a whole new document type.

If a role definition and, presumably, the name chosen for the role, make sense as a reflection of an actual role in the office, the role should be comfortably usable without technical knowledge of its definition. Most end-users would simply use these fixed roles and assign people to them when defining their working relationships with particular individuals. Thus, a manager could indicate that Harry Smith is acting in the "secretary" role for all of his business data without delving into the details of the definition. Harry would then be granted certain default access to the manager's data and application programs. For example, in the manager's calendar Harry may be able to read all of the business appointments and to add appointments during business hours. In the budget data base, Harry may be able to read the accounts of all of the manager's groups without seeing salary information for the senior personnel.

Any details which do not suit a given office and the way one manager chooses to work with his secretary can be changed by *modifying* the standard role definition. For example, in the calendar one could suppress notification of new appointments added by the secretary if they occur during certain office hours. For the office worker who establishes a totally unprecedented mode of working with others, the primitives of the system can be made available so that he can build a full role definition, although likely as not designing a complete new role definition might be left to an expert advisor (as is

the intention in the SCRIBE formatting system).

5. Conclusion

Computer support of cooperative activities requires that information about the "players" be an integrated part of the system. This includes information about dynamically changing roles in real-time meetings, as well as long-term working relationships.

Roles and working relationship information bears some relationship to access rights and locking protocols seen already in data bases and operating systems. The notions are different in that roles are expressed in terms that are meaningful to the organization. We expect the office workstation to provide certain generic role definitions such as secretary, manager, author, editor. Also, alternative meeting support packages will provide roles for different meeting structures. These include the notions of *chairman*, *controller* (person with the floor), and *participant*, and appropriate operations for each, such as requesting the floor and passing control, as well as certain voting procedures. A major component of our research is the continuing study of offices, office work and meetings, in order to identify those roles and working relationships which should be provided.

The ideas expressed in this paper are the basis for an ongoing research project. The goal of our research on cooperative office work is a coherent design for cooperative office systems embodied in a set of software modules for an office workstation. The interfaces to all end-user packages should be consistent, all packages would allow for user tailoring of the work environment and in particular all cooperative work should be supported through a uniform interface for defining roles and working relationships.

6. Acknowledgements

The research described in this paper is being funded by the Advanced Research Projects Agency of the Department of Defense. I am grateful to Dr. Marvin Sirbu, Sunil Sarin, John Cimral, Stan Zdonik and Albert R. Meyer for their comments and suggestions about the work described here.

7. References

Good, M. 1981 "ETUDE and the Folklore of User Interface Design" MIT Office Automation Group Memo OAM-030. Massachusetts Institute of Technology. Cambridge, Mass. March, 1981.

Good, M.D. 1982. An Ease of Use Evaluation of an Integrated Document Processing System.

Proceedings of Human Factors in Computer Systems. Gaithersburg, Maryland. March 15-17, 1982.

Greif, I. 1981. PCAL: A Personal Calendar, MIT Laboratory for Computer Science Technical Memo, TM-213. Massachusetts Institute of Technology. Cambridge, Massachusetts. December, 1981

Greif, I. 1982. "Teleconferencing and the Computer-Based Office Workstation." Proceedings of Teleconferencing and Interactive Media. Madison, Wisconsin. May, 1982.

Hammer, M., R. Ilson, et al. 1981. "Etude: An Integrated Document Processing System." Proceedings of the 1981 Office Automation Conference. AFIPS. March, 1981.

Kunin, J. S. 1982. Analysis and Specification of Office Procedures. PhD Thesis, MIT, Department of Electrical Engineering and Computer Science. Massachusetts Institutes of Technology. Cambridge, Massachusetts. January, 1982.

Reid, B. K. 1980. "A High Level Approach to Document Formatting," Proceedings of the Seventh Annual Symposium on Principles of Programming Languages, January, 1980.

Sirbu, M. and S. Schoichet, J. S. Kunin, M. Hammer, J. Sutherland. 1982. "OAM: An Office Analysis Methodology." Proceedings of the Office Automation Conference, San Francisco, California. April, 1982.

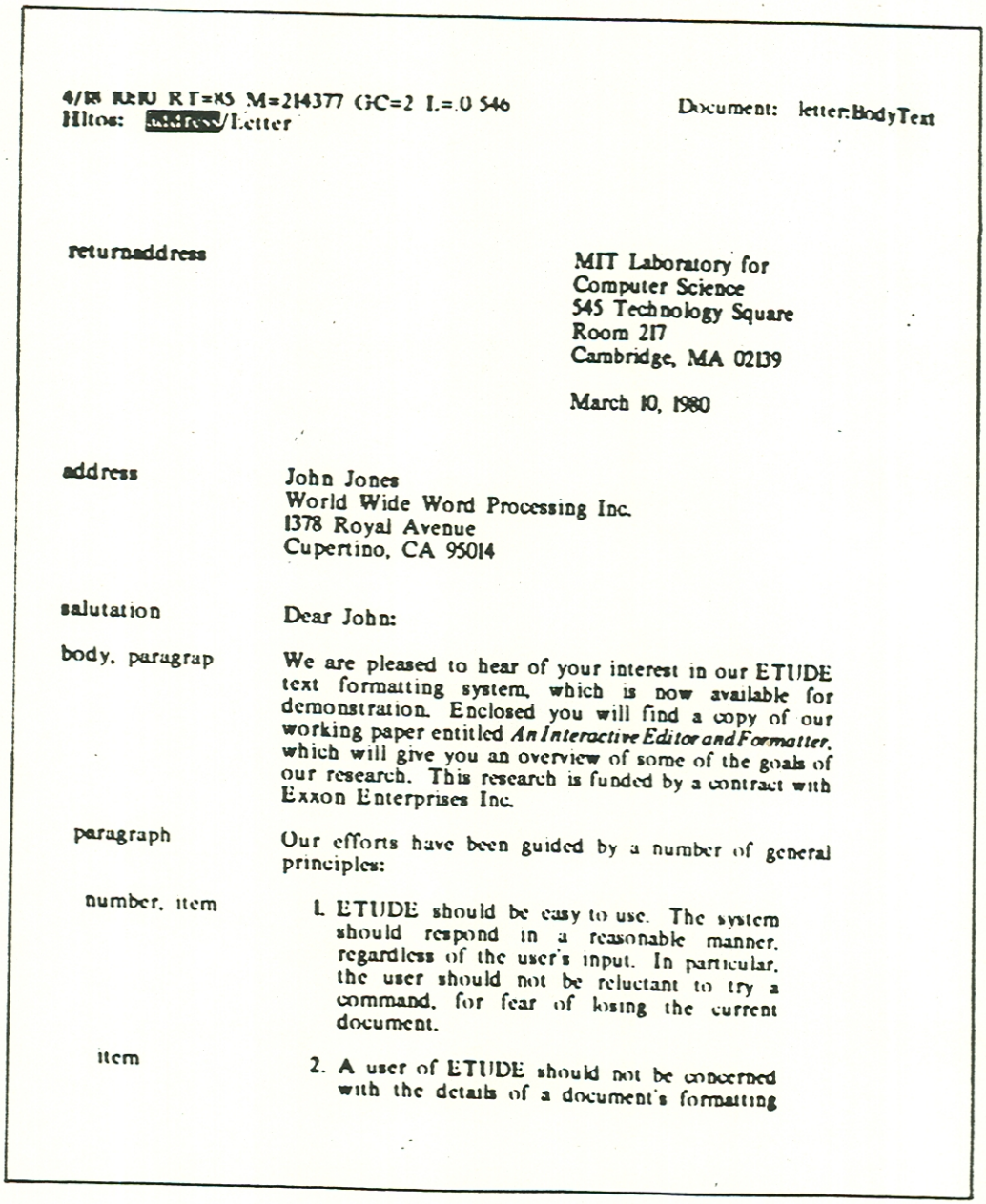

Figure 1: The ETUDE screen contains a letter in the main display, formatted as it will appear when printed. Along the left side of the screen is a summary of the logical structure of the letter. The status window at the top of the screen indicates that the document is a letter (top right) and that the user's cursor is in the address component of the letter (highlighted in upper left hand corner).

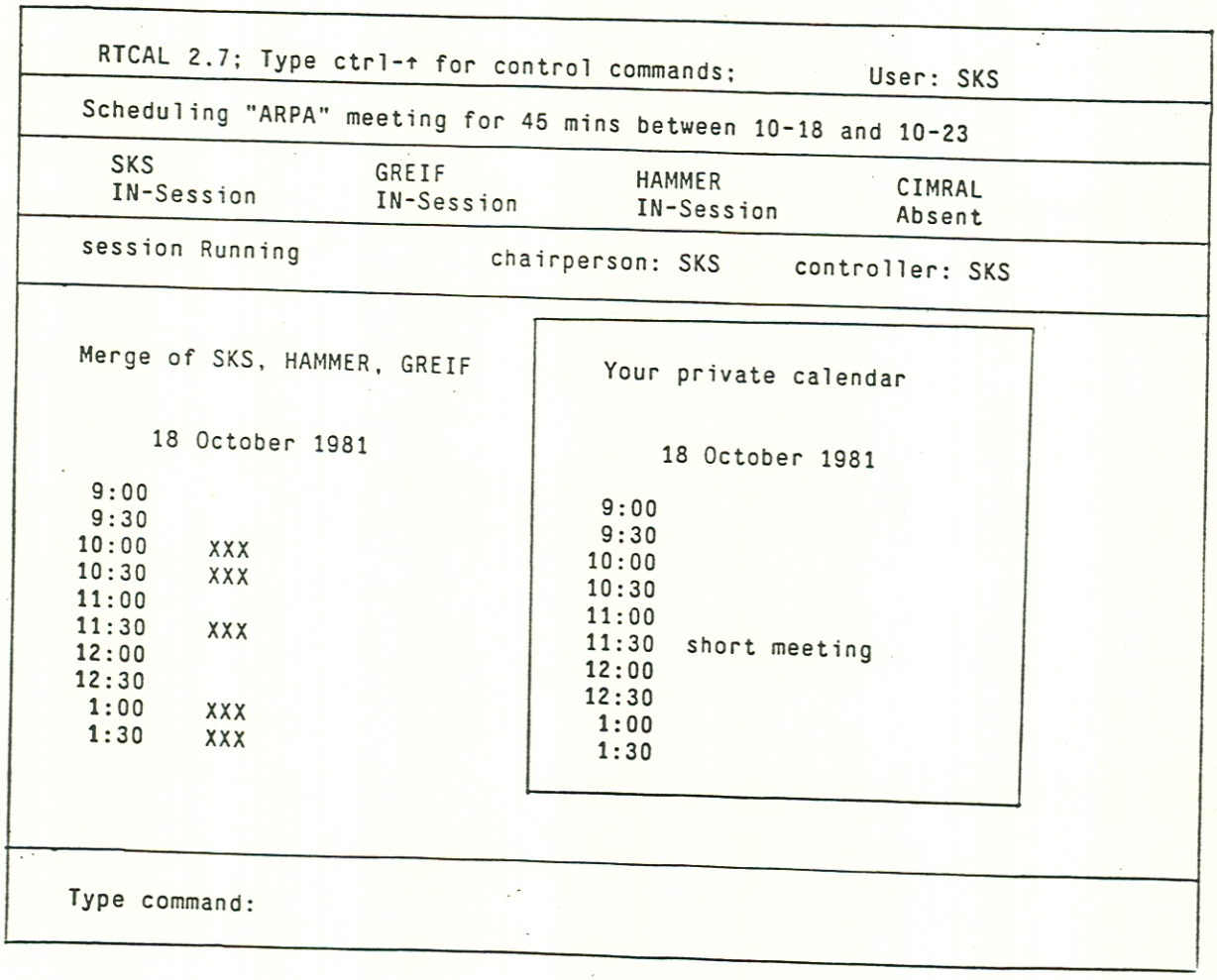

Figure 2: The display of RTCAL presents both shared and private workspaces. In this case the session has been called to schedule an "ARPA" meeting. The session is "Running" after three of the four invited people agree to participate. Their schedule cards are merged and displayed in the shared space to the left. The screen is shown from the point of view of "SKS" (Sunil Sarin) so that his private calendar is displayed to the right.