RC 8384 (#36518) 7/30/80
Computer Science 39 pages

Office

Research Report

PICTUREWORLD: A Concept for Future Office Systems

W. Schild
L. R. Power
M. Karnaugh

IBM Thomas J. Watson Research Center
Yorktown Heights, N.Y. 10598

IBM Research Division
San Jose · Yorktown · Zurich

Copies may be requested from:
IBM Thomas J. Watson Research Center
Distribution Services 36-068
Post Office Box 218
Yorktown Heights, New York 10598

RC 8384 (#36518) 7/30/80
Computer Science 39 pages

PICTUREWORLD: A Concept for Future Office Systems

W. Schild *
L. R. Power
M. Karnaugh

IBM Thomas J. Watson Research Center
Yorktown Heights, N.Y. 10598

ABSTRACT: After a characterization of the work of office principals and a statement of some qualitative requirements, the system concept called "Pictureworld" is described. Pictureworld provides the user with an understandable, self-prompting interface which displays iconic representations of familiar office objects. It is easy to learn and to remember how to command the system. Functions for an entry level system are defined. A software organization in terms of Pictureworld objects will facilitate maintenance and expansion of the system. In view of large areas of ignorance concerning the acceptability and value of principal support systems, Pictureworld is proposed only as one component of a larger program of research.

* W. Schild was formerly with IBM Israel.

1. Introduction

The anticipated widespread use of electronic systems to support the work of office principals will bring a very large group of non-programmers into frequent close interaction with information processing systems.

Because the users will not be highly motivated to develop their programming skills, the design of principal support systems poses some special problems. Either the services offered must be stereotyped, narrow, and inflexible or else the user interface and the system software must provide much more competence in modeling the users' task environment and needs than has previously been attempted.

The office environment already provides support devices that aid principals in stereotyped, narrow, inflexible and also non-integrated ways. These include the telephone, the file cabinet, and the hand calculator. The challenge faced by the information systems designer is to bring into being a class of systems that come much closer to meeting the principals' needs. This is not merely a question of good implementation; the functional requirements and cost-effectiveness tradeoffs are not yet well understood.

This report includes a discussion of the principal and his work (Section 2), some of the qualitative functional requirements for effective principal support (Section 3), and in later sections, an approach to the designs of the user interface and the system software for a proposed experimental system. This approach will be called "Pictureworld."

We have not implemented a Pictureworld system and we make no claims for it as a potential product. We do believe that its implementation and test would offer useful new knowledge about such products. This is not a vacuous claim. It is entirely possible to build and test systems without adding much to our prior knowledge, for example, that system X is much too limited or that system Y is much too complicated. A test system must charm a representative group of test subjects into using it intensively if we are to learn really interesting things about its potential and its deficiencies.

2. The Principal and His Work

The term "principal" will denote a person who works in an office and whose productivity is, or could be, substantially enhanced by secretarial support. Principals' roles include manager, administrator, and professional; and they are found in all major industries, in government, and in education, law, and medicine. We specifically exclude office workers who perform highly structured, repetitive tasks such as bank tellers, reservations clerks, and word processors. Non-principals may, however, have access to principal support systems in supporting roles.

We believe that the most important goal of a principal support system is to increase the principals' productivity and to improve the quality of their work. The displacement of supporting personnel, such as secretaries, clerks, and typists, is regarded as an interesting side effect. Another possible side effect is to upgrade the skills of supporting personnel. Because of these factors, it is more difficult to measure the value of principal support systems than, for example, systems that improve secretarial productivity. The productivity of many principals is difficult to measure. We shall not deal explicitly with this problem.

Principals are distributed across all industries and in a variety of roles. In order to lend substance to the concept of principal support system, it is necessary for us to identify a common group of tasks and needs. This will also provide the basis for a tentative, qualitative specification of system requirements.

Our understanding of principals and their needs can be aided by considering the supporting role of the secretary. Secretarial services commonly cover three broad areas: communications (e.g., typing, copying, and distributing documents, taking telephone messages, mail handling), filing and retrieval (of documents or data), and scheduling (e.g., calendar maintenance, making appointments, scheduling and issuing reminders). Some services cross these boundaries. For example, keeping distribution lists is a filing function that aids in communications. We are more interested in seeing what is typical than in rigorously partitioning the class of services.

A few more generalities will aid us in understanding the work of principals. We have already noted that it is relatively unstructured, which is to say, varied. Their tasks may require decision making and negotiation; and they may be of long duration. Because of the needs for intermittent communication and fact finding, principals' work is frequently interrupted. Also many tasks may be simultaneously pending. These observations also point to common and pervasive needs for aids to communication,

information retrieval, scheduling, and the organization of work.

A principal may be expected to take pride in his dealings with other principals, in his expertise in some special field, or in macroscopic evidence of the results of his work, such as sales volume, cases successfully handled, etc. He is less likely to be motivated by the development of supporting skills, such a learning the intricacies of his firm's computing system. This may be contrasted with the pride a secretary might take in such skills as typing, shorthand, or filing. Furthermore, a principal will devote less time to the exercise of such skills.

Despite these generalities, we wish to avoid certain commonplaces about principals which may turn out to depend strongly upon support system implementation and office sociology, neither of which is immutable. Questions concerning principals' willingness to accept automation, display terminals, keyboards, writing tablets, or changing modes of communication are not properly answered by means of introspective judgments, nor will attitude sampling suffice. The results will be valid, at best, only in the existing office environment. Even trials of new systems can be misleading unless the subjects are properly motivated, the system being tried is well implemented, the scope of services offered is adequate, and the experimenters are very scrupulous in the experimental design, the data gathering, and interpretation of the results.

3. Some Qualitative Requirements

Having surveyed the general characteristics of principals' work and categorized the secretarial services commonly received by principals, we are able to assert that an integrated principal support system should aid the principal in the following ways:

* making communications quicker, easier, more reliable,
* giving the principal quicker and easier access to documents and data,
* aiding the principal in organizing and scheduling his work,
* providing reminders, status summaries, and
* providing guidance for relatively standard tasks.

In addition, because of the variety of principals' roles and industries, the system should be expansible to permit the addition of specialized applications or to provide access through the system to pre-existing software services. Also, user requirements for data sharing, data privacy, and data security must be satisfied.

Finally, we come to the all important question of the user-machine interface. This includes the function and appearance of the terminal I/O equipment, system protocols, the control language, system responses, etc. It also may be considered to include self-teaching and help facilities of the system. Prior to experiment, we can offer only some generalities on these questions. These are aimed at making the system initially acceptable to principals. After some experience with the system, users may be ready to accept greater complexity and further change; but such details are even more speculative.

We wish to minimize the user's entry cost to the system (i.e, the effort required to learn to use it). In addition, since the system will be used intermittently rather than continuously, the system commands should be simple, natural, and easy to remember. One way to achieve these things is to make the system interface resemble the normal office environment. An interface which prompts the user in natural and understandable ways is also desirable.

Another desirable feature of the interface is to interfere as little as possible with the cognitive processes of the user, so as to maintain high productivity. It is worth keeping in mind that, when a user is working at a terminal, he is timesharing two cognitive activities. One of these is the task he is attempting to perform. At a principal support terminal, this will typically be the reading, composition, retrieval, or disposition of some document or data. The second activity is to interact with the system in order to obtain the desired services. These two activities

clearly interfere with one another by competing for the user's attention.

If a user is forced to contemplate the cleverness of the system's designers each time he tries to recall the rigid protocols of his next supplication to the system, his productive work is delayed. If delayed for more than a few seconds, the current state of his work will have vanished from short term memory and will need to be restored before he can continue.

Even when the control language is familiar to the user, some cognitive interference will take place. Activities which require the same cognitive faculties usually exhibit more mutual interference than more divergent activities. This is common experience. For example, it is difficult to carry on two conversations simultaneously. On the other hand, excepting emergency maneuvers, one can easily drive a car and converse with a passenger. That is, the visual and motor activities used in driving interfere very little with generating or hearing spoken utterances. There is a body of work relevant to such interference in the field of cognitive psychology (Kintsch). Paivio has proposed a "dual coding theory" of memory in which memory of images and verbal memory are separately encoded. Brooks has shown that reading interferes more with the simultaneous representation of spatial relationships, while listening interferes more with verbal memory. Atwood has shown that visual tasks interfere more with recall of imaginal phrases such as "nudist devouring a bird" whereas auditory tasks interfere more with recall of abstract phrases such as "the intellect of Einstein was a miracle".

In using iconic representations of familiar office objects (on its display) as a control interface, the Pictureworld concept achieves familiarity, naturalness, self-prompting, and (plausibly) minimal cognitive interference with the primary task. That is, the primary task is typically verbal while the control task will be primarily imaginal (i.e., the recognition of icons) and motor (i.e., touching the display panel). The character of this interface is more fully explained in the next section.

The Pictureworld concept, as described here, deals with a circumscribed set of "core" functions for principal support. Our emphasis is on those aspects of the interface which will make it natural and acceptable to the general user, and on a software organization which will be maintainable and easily expansible. We wish to give the principal direct control of an office environment which functions according to his own understanding of it, and which can be augmented according to his special needs.

4. The Interface: Pictureworld

We suppose the principal to have access to a high performance graphics display device. The display ideally would have a touch sensitive panel overlaying it (though this is not essential) and a writing tablet or keyboard for input. The screen of the display device will exhibit all user and system actions both potential and actual. The primary source of communication is a set of graphic icons of office objects and related symbols. Figure 1 shows a representative set for purposes of illustration -- a complete set is given below. As items are selected (e.g., by touching), they are highlighted. Touching a second time turns the highlighting off thereby deselecting the object. These and similar conventions are used to establish protocols that insure user and system understand each other. The result of any selection is a command to the system which, when executed, causes a change in the screen's appearance as, for example, a blow up with full details of the calendar page for a given day. A command may cause a change internally in the system as well.

All actions by the user are performed by manipulation of one or more objects or their parts. Entry of text data cannot, however, be accomplish this way and will be performed either through a keyboard or via tablet. The result of specific actions may cause certain changes in the object's appearance. In addition an object will expand or contract on the screen as it becomes or ceases to be the focus of attention. Options are indicated on the screen in picture form where possible, thus for example the source and destination of a document will be indicated by an arrow. Editing of text via tablet can be done by mimicking (roughly) paper and pencil proofreading conventions where the symbols used represent well known pencil marks. Certain (function) "buttons" may be defined to allow greater flexibility, since icons for these would not enhance the interface and thus serve no useful purpose. Examples of these include "do", "undo", and "clear".

An important aspect of this Pictureworld approach is that one can easily distinguish between commands and data without needing to be conscious of it. We thus overcome a traditional stumbling block of most interfaces. Furthermore, suspending a particular activity (e.g. looking at one's mail) requires no special user action and no formal protocol to return. Where the screen size limitations require a certain amount of housekeeping, we provide techniques to achieve this with minimum perturbation. We maintain a single level of interaction with a global context (the "office") constantly in view, thereby minimizing backtracking.

5. Sample Scenario

In order to illustrate the approach we present a sample scenario. We assume the existence of a host operating system with communication and storage management facilities. Specifically, we hypothesize basic file support and electronic mail facilities like those available under VM/CMS.

A principal, having logged on to the system, is shown a basic view of his office as in Figure 1. A glance at his IN-OUT box shows the existence of incoming mail (depicted by a small envelope in the IN portion.) He decides to scan his mail and selects the IN portion (by a touch or cursor movement followed by "do"). The screen appears as in Figure 2 showing the various letters with dates and sources. The first letter of interest is from A.B.C. Corp. and this one is selected and placed on the desk by touching the arrow pointing to the desk. Other arrows shown reflect options in case of other destinations. (In fact, we envisage having various kinds of arrows -- e.g., move, copy).

The letter now appears on the desk as in Figure 3. The IN box retains a picture of an envelope since there is additional correspondence to be dealt with. In keeping with the spirit of using real world analogues, we make use of the concept of physical pages to be manipulated. Thus, rather than scrolling, the principal turns pages forward or backward. In Figure 3 for example a selection of the lower right corner of page 2 would indicate a desire to view the next page. Actions that involve real world analogues (moving documents, viewing them, addressing mail) are executed "on" the desk. The significance of this will be discussed below.

We continue with the sample scenario. Having read the letter, the principal notes that it requests information and a document (an invoice). He therefore selects the file cabinet indicating to the system that something is to be retrieved. The next screen appears (Figure 4) with a "prompting" file folder for appropriate search parameters. These are entered as desired. (We discuss data base and retrieval considerations later.) With the template file folder completed, the user indicates he wants to execute the search by pressing the "do" button. The result is as shown in Figure 5. Notice that the desk has receded but an icon on it reminds the user that a document is still on it. The result of the search yields a number of matches as depicted on the screen in the page marked "INDEX".

The creation of this index by the system illustrates how we adjust the electronic medium to accomplish a useful result in a manner not unlike what a secretary might do. That is to say, as a consequence of the search, the system

constructs a document listing the possible entries meeting the search request. This document may be referenced like any other (e.g., filed, mailed) in addition to acting as a reference prompt for selecting the item of interest. Suppose the user now selects item 2 to be placed on his desk. The screen then appears as in Figure 6. Because the desk previously had the original letter on it and space limitations precluded a full display of both documents, the solution taken shows the letter moved (by the system) to the upper right corner in a "pending" box. This sort of automatic housekeeping is felt to be acceptable since the results are visually presented. We note that one of the goals of the interface is to have all actions and reactions explicitly represented on the screen. At present, we conceive of the pending box to be a lifo stack which can be "pushed" or "popped" either by the system or by the user. Alternatively it could be modeled as a "temporary" file cabinet and searched in a similar fashion.

Having verified that the document is the one desired, the principal now wishes to transmit it and therefore selects the envelope on the left margin of the screen. The resultant screen is depicted in Figure 7. The envelope appears on the desk (where all mail-related activities take place) and the cursor points at the addressee for entry of the appropriate information. A "smart" system might fill in the address once the name was given, or better yet, surmise that the recipient might be the same as the source of the last letter scanned. The figure also shows that greater flexibility can be achieved if one allows different sizes of the same object depending on context. In this way more than one object can, for example, occupy the desktop (albeit with reduced information content) and the system achieves greater expressive power in the manipulation of documents.

The invoice is placed inside the envelope by selecting the arrow joining the two and then the memo pad on the right margin is selected so as to initiate memo creation. Figure 8 shows the result of these two actions. We now have a form to be filled in by the user with appropriate text. Note that the envelope now reflects previous actions and, since it now contains a document, the destination option, as indicated by the arrow to the OUT box, is shown. When the memo has been completed, it too is inserted in the envelope with the result shown in Figure 9. At this point the letter in the pending box could have returned to its previous position in the center of the desk automatically. For purposes of exposition we show the alternative in which the user would manipulate the contents of the pending box.

The letter is now transmitted by selecting the arrow to the OUT box with the result shown in Figure 10 (note the OUT portion reflects this fact). If the recipient were also on the system, transmission would be effected electronically;

otherwise the appropriate documents could be generated in hard copy. The scenario concludes with the original letter being filed away by source and date and with an option (not shown) for the user to include his own indices.

We have briefly illustrated the primary requirements of the system. It is based on an intuitive real world model, with the actions obvious to the user. Essential aspects are constant feedback, the maintenance of a global context which allows reminding of unfinished work, minimal cognitive interference between command entry and text manipulation, and the minimization of keying. The approach lends itself to extensibility and to a personalized implementation. All these suggest the system would be easy to learn and to remember.

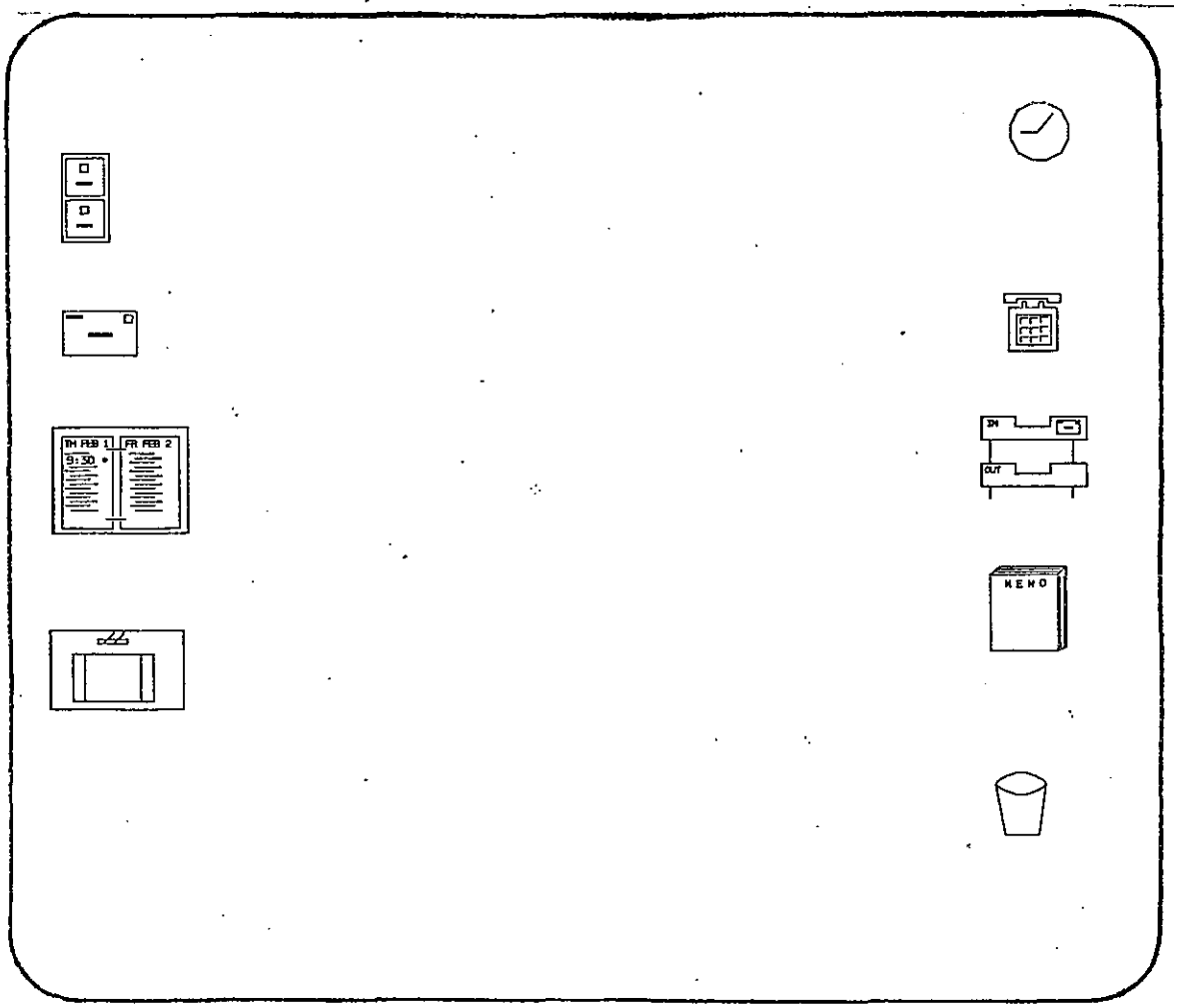

Figure 1 Initial display

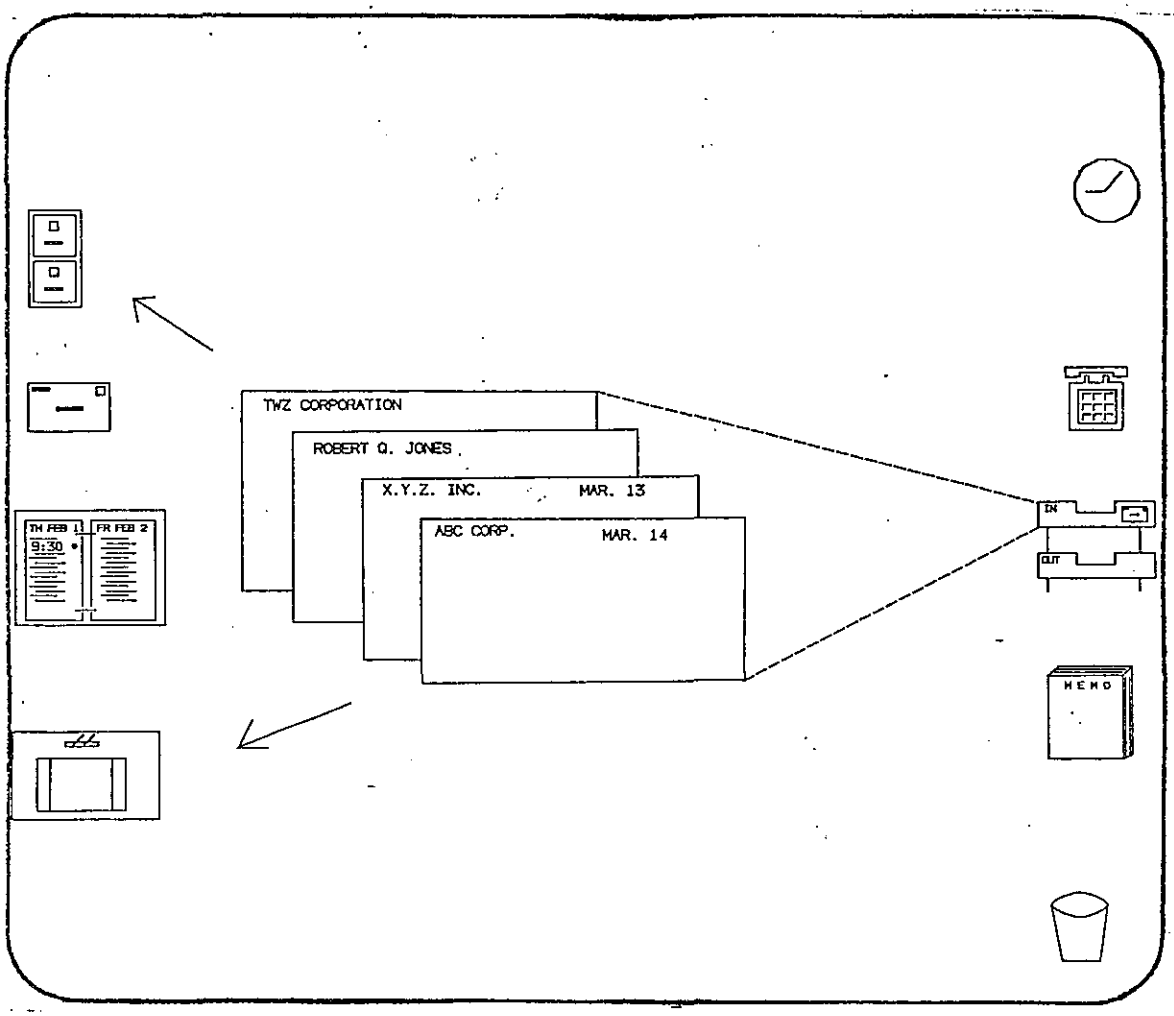

Figure 2 Scanning mail

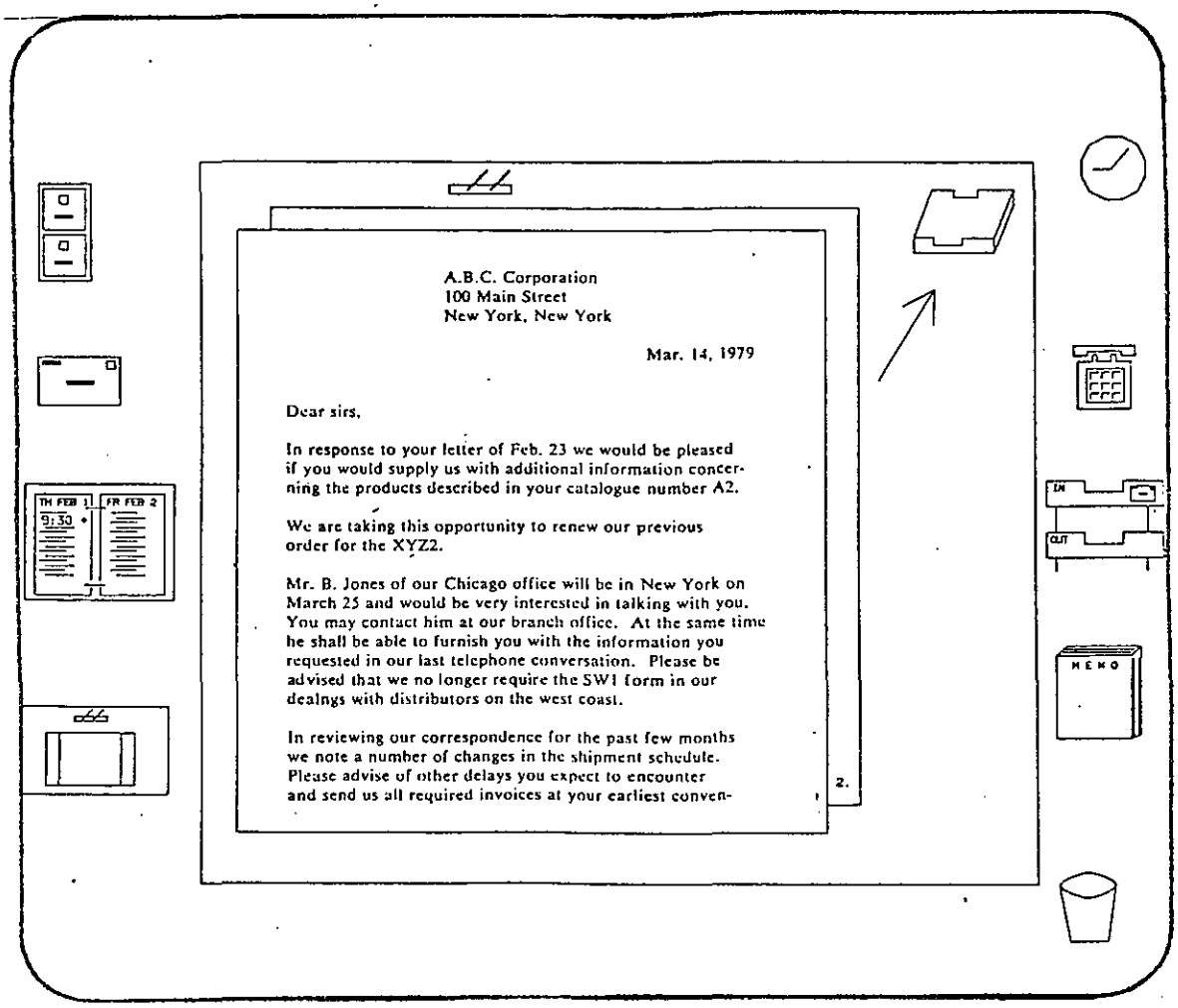

Figure 3 Reading letter

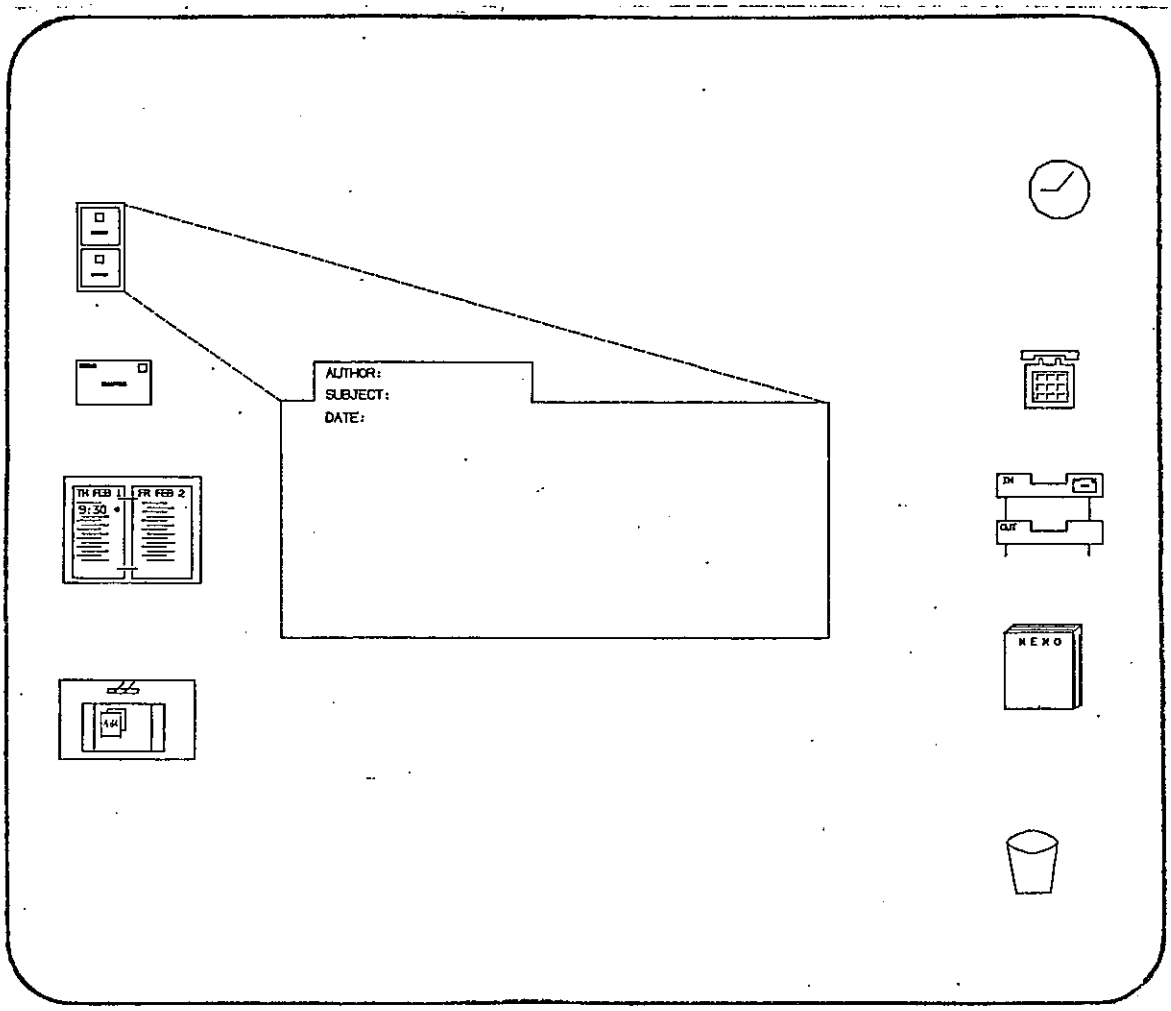

Figure 4 Searching file

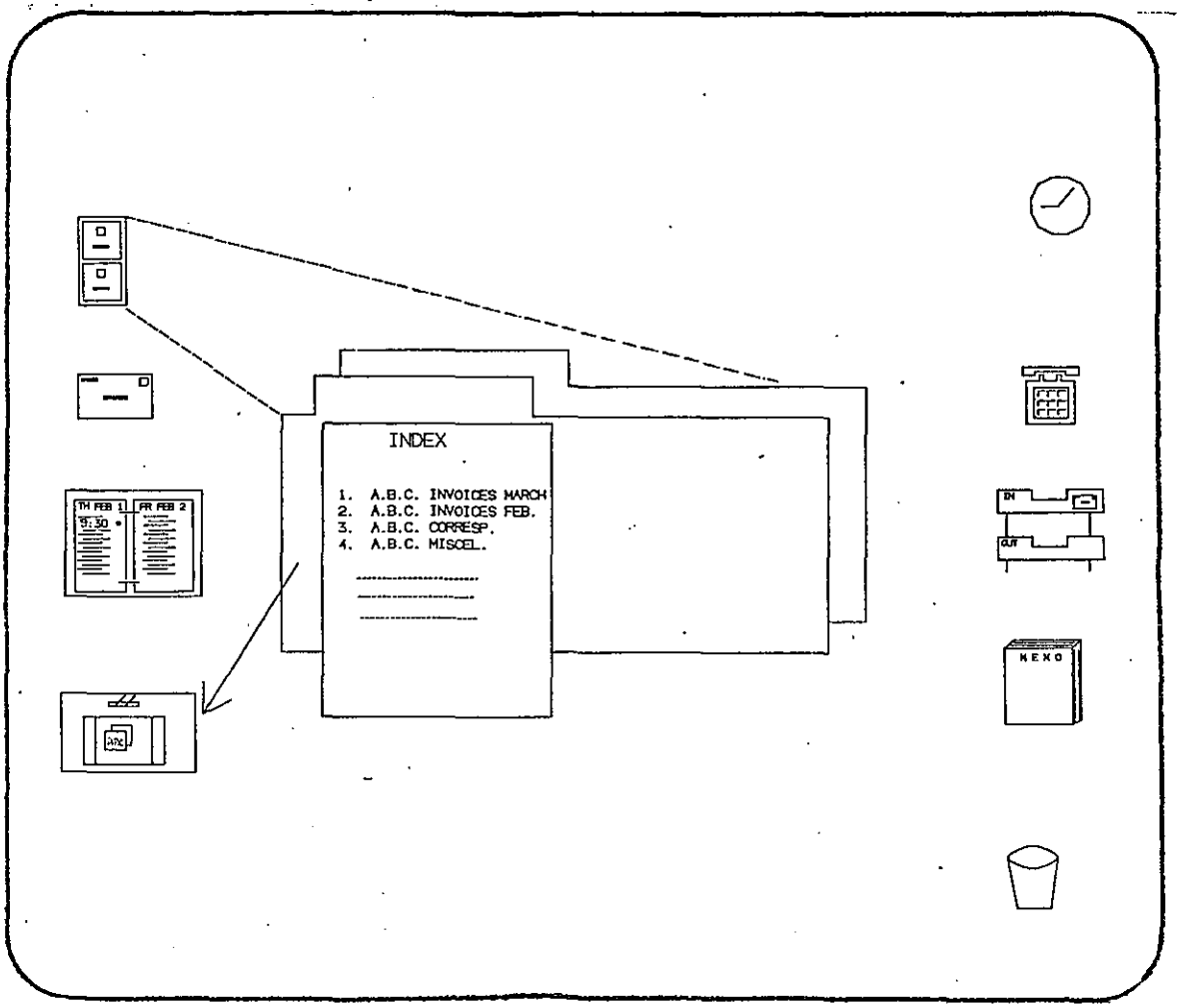

Figure 5 Results of file search

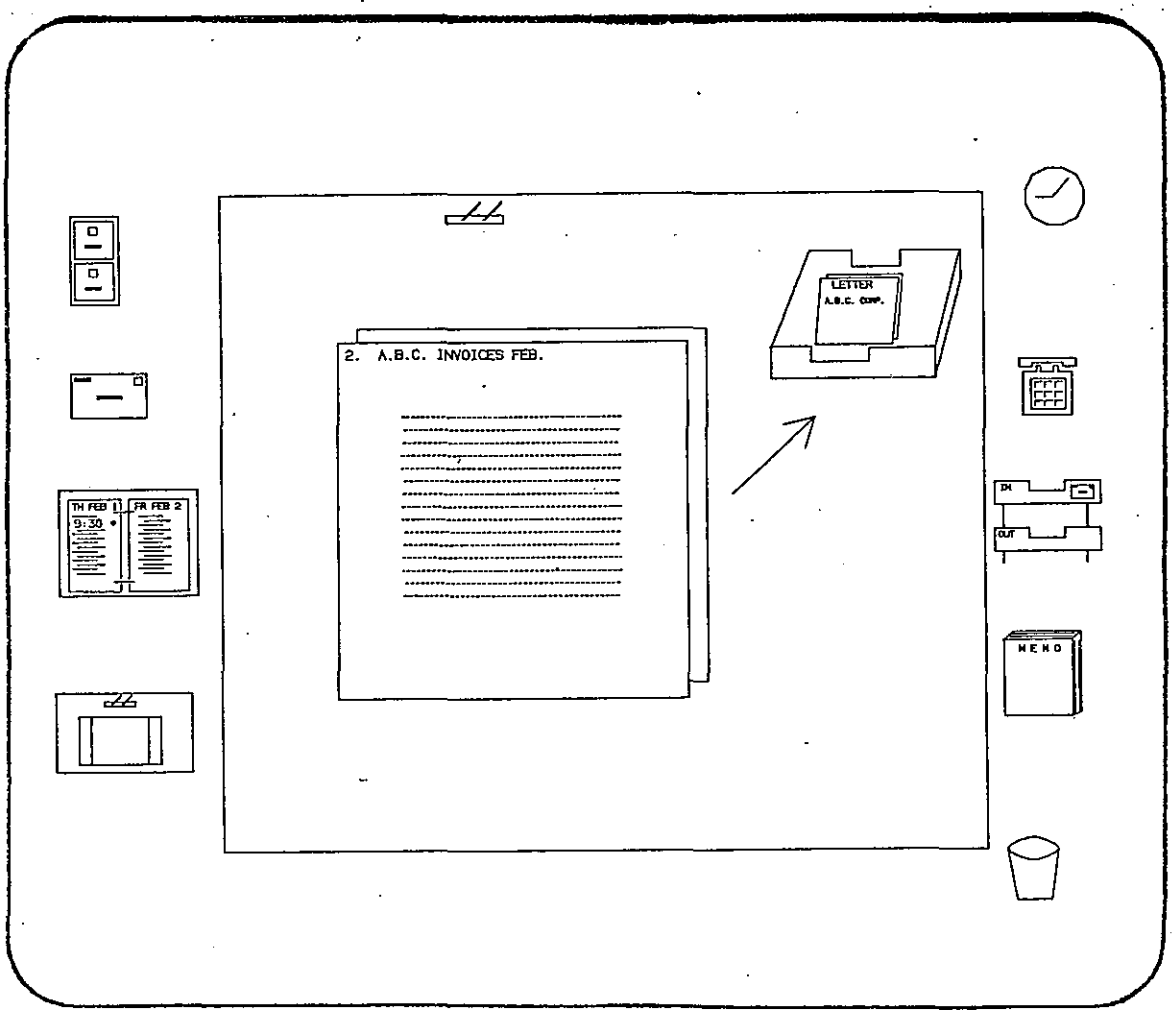

Figure 6 Reviewing retrieved document

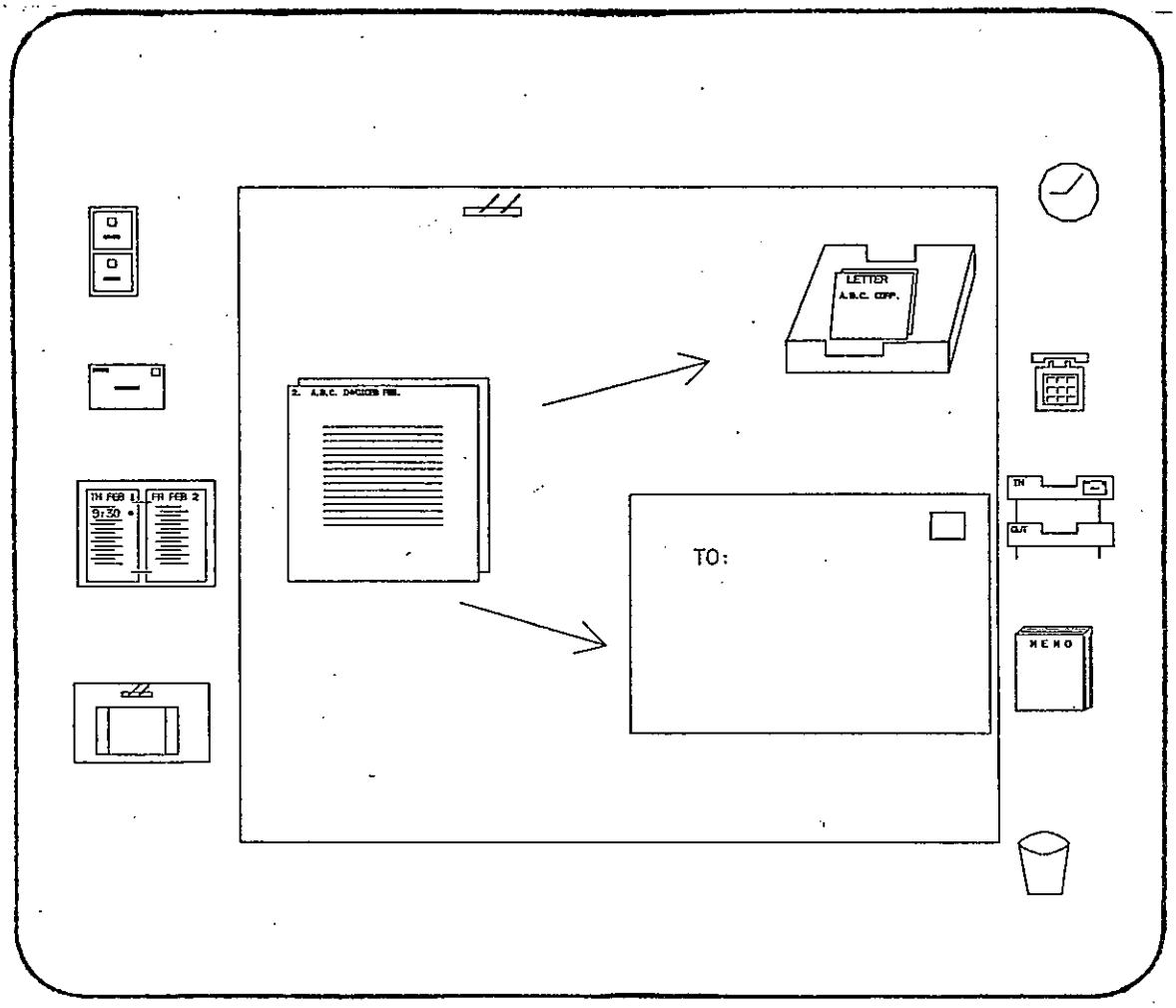

Figure 7 Initiating mail

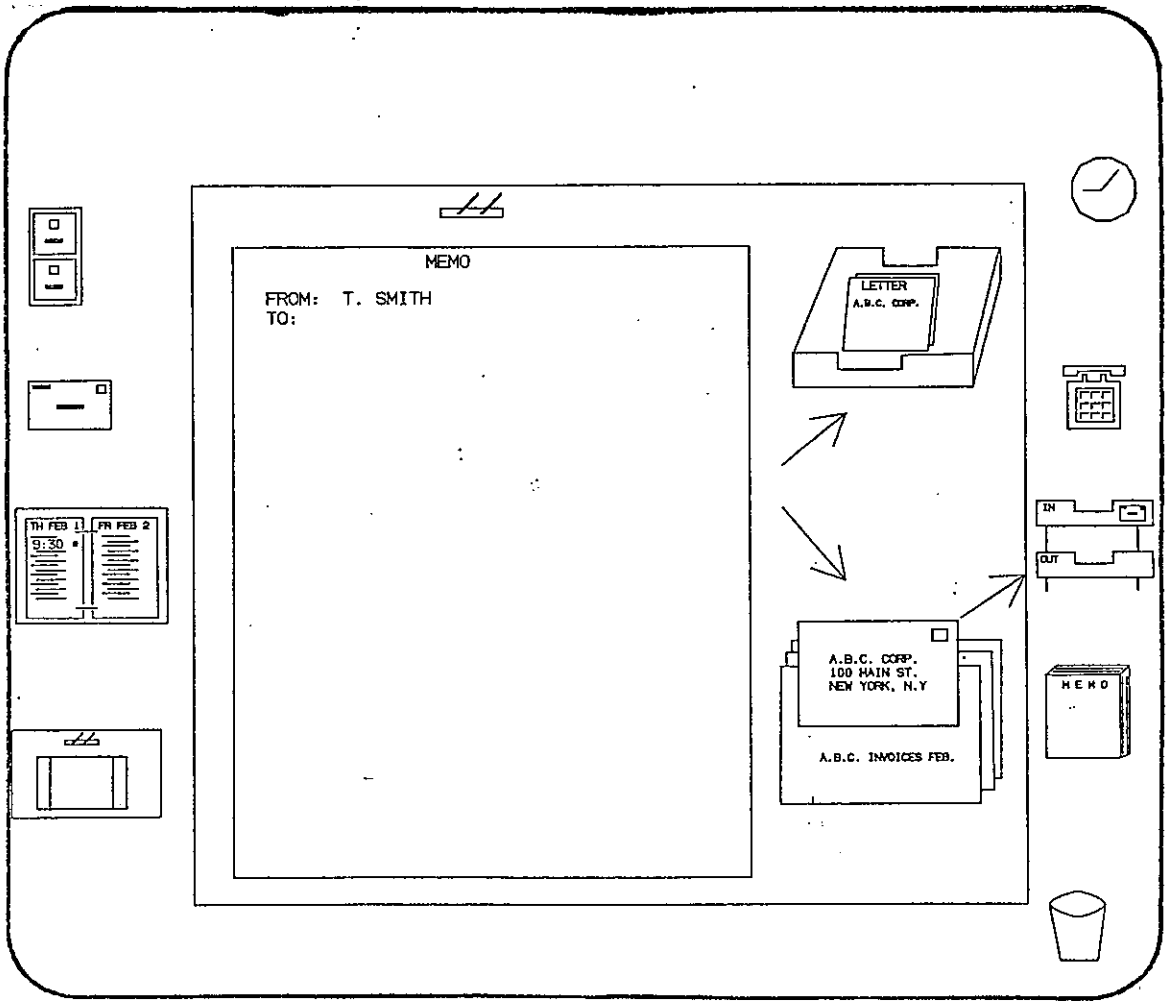

Figure 8 Preparing cover letter

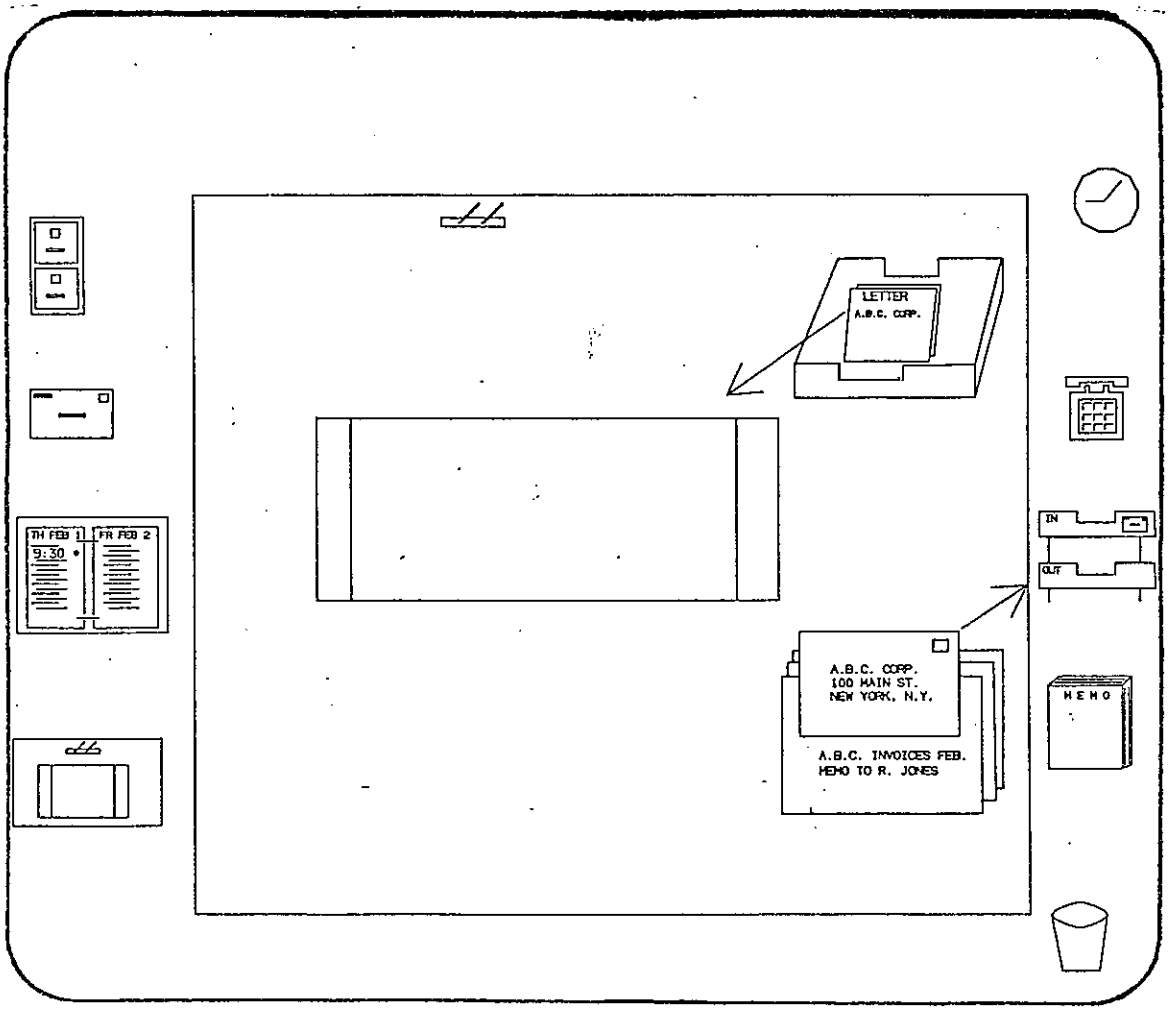

Figure 9 Mailing documents

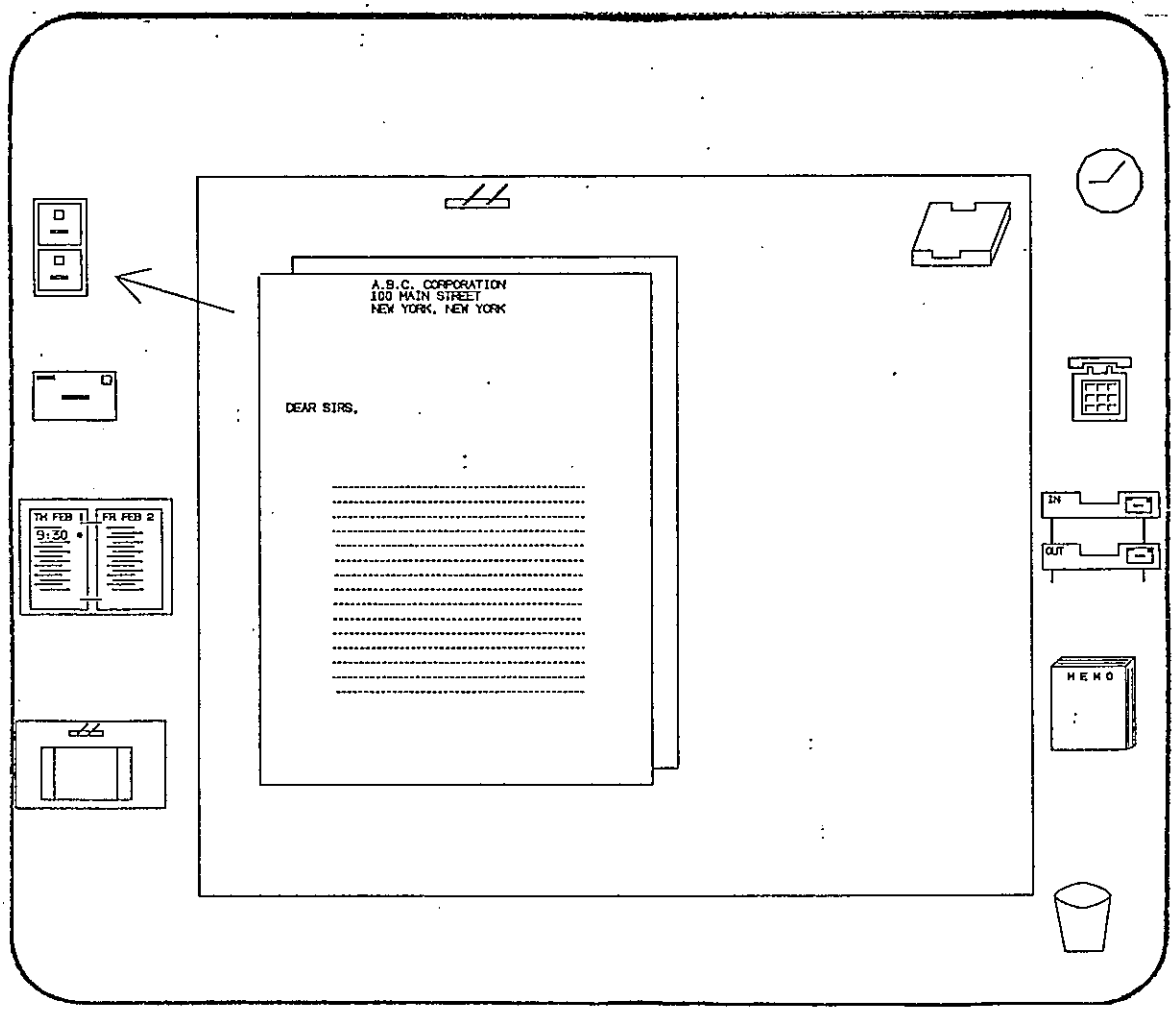

Figure 10 Filing letter

6. Basic Objects and their Attributes

The system is both conceptually and functionally built around the notion of office objects. Each such object is defined in terms of its external icon and user interface, and internally as an abstract entity with associated structure. Each object has its own set of allowed operations and data structure as well as a means of communicating with other objects. An object can contain other objects as, for example, the desk object which contains the pending box. In this section, we focus on the system's functional capabilities by outlining the objects in the basic system and their associated actions.

The basic set was selected because it meets the essential needs of most principals as we have characterized them, and because the objects tend to be interconnected in daily routine. For each object, we present a set of core functions defining the most primitive actions which it supports and a set of extensions which could offer a richer context for implementation.

Objects can fall into a number of classes. The calendar is typical of one that is itself stored, updated and retrieved. Other objects play more passive roles. Thus the envelope object is used to specify the start of a particular activity, namely addressing a piece of mail, and nothing else. (Similarly for the memo pad object.) A third category is represented by the desk object whose prime function is organizational and whose description is based primarily on the fact that it contains other objects -- a pending box, documents, etc.

Calendar: This object will be used to maintain a daily record of activities. The display is of a single day with half-hourly slots. The basic actions supported are: retrieval of a given day by entering it over the current day displayed, leafing through one day at a time by selecting subsequent or previous pages, and updating (entry or modification) of specific time slots. A day is selected simply by changing the day on the calendar and executing the "do" function.

The calendar presents a useful object for expanded function. Two automatic ones that could be implemented are reminders and special cases. Reminders would be implemented by having the calendar call attention to itself through highlighting or blinking when the time corresponding to an entry is near. In the scenario presented, we alluded to this by showing the calendar with an asterisk next to the 9:30 entry. If the principal were, for instance, scanning his mail around 9:15, this entry would light up. If the calendar were then selected, the full page would be shown with the required detailed entry highlighted.

A further extension is envisioned in which the user asks to be reminded. For this case we suppose the calendar to have a reminder option at the top of each page. When this option is chosen followed by the selection of a specific time slot, a reminder similar to the one just mentioned would take effect. Another extension involves retrieval of a given day based on some context. Thus, along with a reminder, the calendar would have a search prompt permitting the user to enter an argument. Finally, the processing of special cases could be built in so that holidays and vacations would be part of the calendar's definition, thereby providing checks when entries were made.

I/O Box: This object serves as the vehicle to handle incoming correspondence and transmission of outgoing items. Both these functions in their primitive form behave much as in the illustration in the sample scenario. Three extensions could improve the usefulness of this object. First we imagine a "prompting envelope" akin to the file folder shown in the scenario which would allow the principal to apply selection criteria to the incoming mail being scanned. A variation of this is to allow priority designation of the incoming mail so that an order of presentation is defined.

The second, and probably most valuable, extension is a means for establishing a distribution on output. One way of implementing this is to have the I/O object recognize a specific document (distribution list) when it is attached to another document for transmission or, alternatively, to scan the document itself for a "cc" list. Another possibility is to define a "distribution" procedure -- this is discussed below. One final extension would be to allow a reminder based on a document's source (or topic) to be set. In this case the I/O box would signal when an incoming mail item satisfying the criterion actually arrived.

File: The filing cabinet object represents our limited response to document storage and retrieval requirements. The system is intended primarily to support personalized files rather than access to large corporate type data bases. Hence the approach has been to devise a facility which permits ready access and storage to a small file store containing recent personal correspondence and related documents. The more extensive file structures involving complex input checks and retrieval searches are not supported directly with this object. For the case of perhaps a few thousand items a classification scheme based on subject, author, and date, should prove adequate. A natural extension would allow the user to specify his own indices for storage and retrieval.

The file cabinet could be extended to more than one "physical" unit allowing classifications of user "files"

thus achieving further personalization of the system. A different icon would then represent each such unit, normally too small to be resolved, and a selection of the file cabinet would cause a prompt among several cabinets, appropriately labeled, to be generated.

When an item is removed from the file cabinet the system would leave an indicator on the appropriate item, indicating it had been removed.

Reference Shelf: This object, not previously referred to, is shown in Figure 11 and illustrates the extendability of the interface as well as its ability to solve special cases. The basic issue is this. Many principals in their daily routine need to refer to certain documents frequently. These may be reference documents, forms, lists, or simply recent incoming items. To retrieve these items quickly one normally keeps them in a handy place rather than a file. The reference shelf object is a separate object containing such items. Retrieval is not done by search arguments, rather selection of the shelf will cause a blow up of its contents, with appropriate labels so that a selection can be made directly resulting in the item being displayed on the desk. As items can be put on or taken off the shelf temporarily or permanently, this object acts as a storage medium (just as the file cabinet). The reference shelf provides a natural repository for directories and, by extension, for directory lookups. Furthermore the use of personal aids such as a spelling checker could be implemented without the use of a separate icon by inclusion within a reference book.

Desk: As indicated earlier the purpose of this object is to serve as a locus of activity involving document manipulation. This is a direct consequence of the design constraint which views the office system as a set of interrelated objects. Specifically, the internal structure forces every object, except the office object, to belong to some other object. This sort of conceptual integrity of the object world is carried all the way through, as will be seen in the office object. The desk is used to display and manipulate documents and associated objects, such as envelopes. A document can be placed on the desk, where it may be scanned (pages leafed through), edited, placed in the pending box, and combined with other documents. To combine two documents we suppose the two to be on the desk (in perhaps reduced form) with a plus sign between them to indicate this option, just as the arrow icon served in the scenario to relate two objects. Moving one document onto another could indicate that they are to be combined and might be reinforced with the visual feedback of the two "clipped" together (see Figure 11). Combining parts of documents is discussed below under editing.

Depending on screen limitations, it may be possible to operate on the desk with two different documents in full size. In any case, by suitable manipulation of the contents of the desk in and out of the pending box, the user should be able to perform most of the document activities he normally requires. The desk object can also function as a kind of scratchpad in order to serve as a repository for informal notes. These notes (actually documents) would not be stored away but be kept in stack fashion on the desk and scanned directly as desired.

Document: This object is the basic source of information. We have already noted the kinds of manipulations supported on individual pages and entire documents. In addition, basic editing and annotation is defined for this object class. In designing the editing function, we explicitly shy away from the standard features common to text editors and incur the loss of generality and power. Our assumption is that the principal will not be inclined to learn conventions that seem unnatural. The solution proposed restricts editing to five notions namely insert, delete, move, copy, and block, which should serve to satisfy the majority of a principal's needs. For deleting, moving, and copying, a block is identified by "encircling" it using the "()" button for the start and finish. The appropriate text would be shown with a box drawn around it. The next button selected would indicate the function performed (delete, move, or copy) and this would be followed by a destination selection in the case of move or copy. (Note that buttons could be physical buttons on the keyboard, or touch points on a display overlay or tablet. If a tablet device were available, the actual symbols for delete could be drawn by the user and recognized by the system.) For insertion the text would be entered and followed by indicating a destination. The system would then indicate the resultant editing on the screen as shown in Figure 12.

All revisions of the document remain indicated until the user requests them to take effect. We can see that an extension of this approach allows indenting to be specified as well as moving blocks of text to other pages. Such moves need not be confined to a single document. We can easily perform a selective combination of documents by movement of blocks of text from one document to another.

If a tablet device were available a further extension that could prove useful would be the addition of an annotation capability. A document will be annotated by entering the text somewhere within the borders of the page. The text need not be recognized and will be stored as is. Should the system include recognition algorithms this could be used to reformat the annotation and present it more cleanly. (The use of a cursive script-like font would be appropriate in identifying annotated material.) All documents would have

default formats. Documents that required more complex formatting would have to be handled by an auxiliary system.

The following objects (included for the sake of completeness) represent no complex user actions and are described briefly.

Memo: Initiates memo creation with suitable form for source and destination prompts. Date is automatically appended.
Envelope: Used to initiate transmission. Serves as object for enclosures. Built in prompt for destination.
Folder: Object used for filing and indexing one or more items.
Waste Basket: Used to indicate document purges from the system. This object could be implemented as a finite or infinite stack, thereby allowing retrieval.
Arrow: This object serves to indicate source and destination of document movement. We envisage at least two instantiations of the arrow, specifically, a move type and a copy type.

Office: All objects are contained within the office object thereby enforcing a conceptual unity to the object world. The function of this object is to represent the global context and to exchange screen management information with the objects it contains.

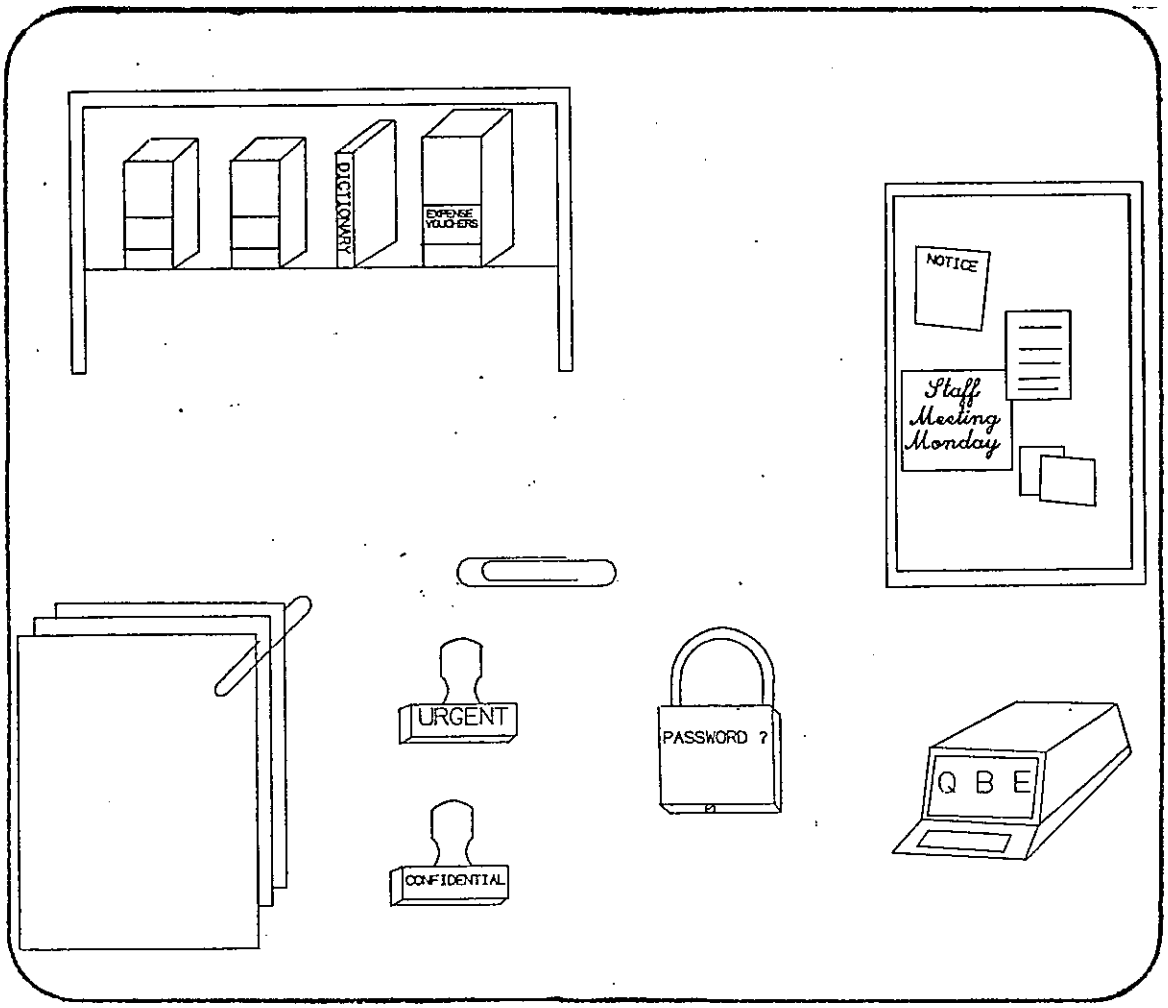

Figure 11 Extended icons

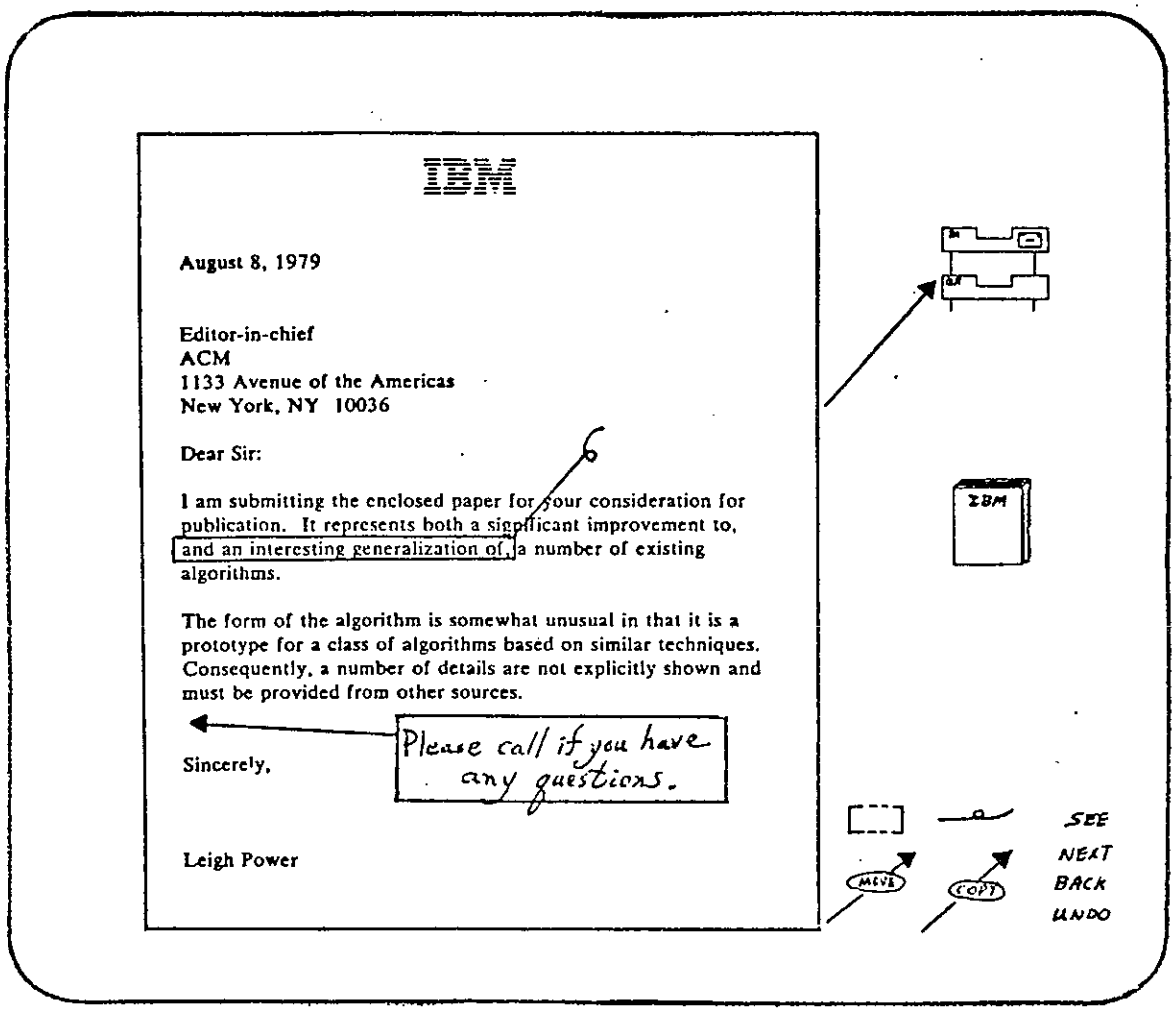

Figure 12 Text modification

7. Extensions

The extensions described below range from straightforward additions to the basic set of objects, to augmenting the system with AI techniques that simulate intelligent task support. At some point along the way between these two extremes the system ceases to be a passive tool and begins to require more sophistication on the part of the user in terms of participation in procedures and understanding the interaction.

The extensions, so far as objects are concerned, involve primarily expanded facilities for the use of documents. We noted earlier that documents in the basic system had no special classification. In an expanded system we propose that the basic document object be instantiated in various ways. Thus we introduce a document type called a list which the user can create or modify. As a particular instance of this class we define a distribution list which has special properties such as being applicable to another document. In a different extension, a document might be classified by applying a label to it. As an example suppose a rubber stamp icon (such as in Figure 11) to be applied to a document thereby classifying it as "confidential", "paid", "original", "on order", etc. In fact, document classes are actually a subset of the more general concept of forms, about which more will be said.

When documents are scanned it is often convenient to be able to "mark" pages. Typically a principal will employ some physical device to do this, bending the sheet, applying a paper clip, etc. In our analogue we suppose a marker icon to be available for placement at any point in a stack of documents. The marker will be visible whenever the document is in full display mode and by referring to it the appropriate page will be selected.

Though most communication in the system is done via documents (memos, letters), it is possible that some principals will find the need for more informal means to transmit audio or written messages, particularly short ones. An obvious model here is the intercom or telephone. A facility to communicate such messages could be a useful extension. Its chief merit would be that it required little formality, and it would be executed immediately. As an example the principal might point to an intercom icon and then simply enter a name followed by a short message. Should there be a few fixed targets of such communications one could define a suitable icon (e.g. intercom with labeled "switches") and thereby further simplify the interface. The possibilities for disseminating information can take many forms. In Figure 11 we show a bulletin board accessible to all users, on which notices could be "placed".

An interactive system is greatly enhanced if the user can undo particular actions. In many cases this feature becomes a double edged sword since the results are sometimes unpredictable and, even worse, not explicitly shown. The interface described here can overcome these barriers with a suitably implemented "undo" feature. Besides showing pictorially the results after the undo button is pressed one could build in an "undo anticipated" function. When chosen, with a particular object and action, this feature would first exhibit the assumed result of the undo, analogous to rolling back a frame of film. The user could then decide whether to have the undo take effect or not. See Figure 13.

As a general rule, the system is meant to be used in standalone mode. However, for the principal who has other application software that he might use in the course of his work, it is easy to provide a smooth transition. Thus the global office could contain an icon representing a QBE terminal which, when selected, would cause the user's terminal to change appropriately. See Figure 11. What should emerge from this discussion is that our particular approach lends itself to significant expansion without impacting the user's learning curve or his ability to function efficiently with the system.

Procedures: While the present design does give a good deal of support for principals, it falls short in providing a means of expressing complex actions, particularly repetitive ones that involve variables -- i.e. procedures. We should like a protocol that permits the principal to both define and invoke a procedure in a natural way. These requirements are difficult to satisfy. All we can do is sketch out, very tentatively, the possible outlines of a solution, yet one that is in keeping with the style and flavor of our approach.

A procedure will be thought of as a scenario of user interactions with the system as depicted by a sequence of individual screens. The sequence of screens by itself can serve to identify the basic flow of processing and the components (objects) that participate. Thus we envisage a user defining a procedure by describing it through a sort of stepping through process. In addition, certain elements will have to be specified explicitly such as parameters, repeating actions, and decision points. In order to minimize a user specification language we suggest that the system be programmed to infer from the sequence the processing desired. This might be achieved by considering carefully the individual screens and their incremental changes so as to deduce the procedure's intent. A generalized icon representing any procedure can be defined so that a procedure could refer to (i.e call) other procedures.

In Figure 14 we hint at the way this feature might look. The frames are meant to show the user how the procedure is understood by the system (prior to accepting the definition.) The example illustrates a procedure for processing new PhD applicants for employment. The sequence entails verifying that the incoming mail item meets the input test for the procedure. If so, the first page (containing the resume) is copied and placed in the pending box. Following this a particular form is selected from the shelf and combined with the entire application which is then circulated to a group specified in a "cc" list. The distribution itself is another procedure (Figure 15) called by choosing the procedure icon and giving its name. (Every procedure icon represents input, process, and output.)

The execution of the procedure by the system would not necessarily entail exhibiting each frame that defined the procedure. Only frames where a prompt for parametric values was required would generally result in an appropriate display. Indeed one can well imagine an extension in which the procedure is invoked and all required variables are specified at once. This leads us to mention one specific instantiation of procedures which has great application -- the use of forms.

In describing the document object earlier we noted that the much more general concept of a form could be implemented to encompass a wide range of practical applications. To see the significance of this we need only remark that the analogy of the principal performing individual office activities is to be found in numerous application areas where a set of individual tasks can often be represented as a sequence of "filling in of forms". Thus a suitably defined form could have much of the expressive power of a procedure. It could provide a natural and effective means of extending the interface. Clearly much needs to be done in design of this aspect of the system, but the prospects seem to warrant a determined effort.

Artificial Intelligence Augmentation: Artificial intelligence techniques are software techniques by means of which a system can mimic the flexibility of behavior which is allegedly characteristic of natural human intelligence. In addition, artificial intelligence systems can exceed human capabilities in those areas in which computers have certain advantages over humans: processing speed, reliable retrieval of data, larger amounts of working memory, indefatigability, etc.

While the state of the art limits present applications of AI to rather narrow domains of knowledge, there are some possible benefits to be realized from the application of AI techniques to Pictureworld. These would permit the system to aid the user in managing semi-routine tasks, to

anticipate some of his needs for document retrieval, and to explain to the user why it has done what it has done whenever he chooses to ask. AI augmentation is discussed at length in a companion report (Kolodner).

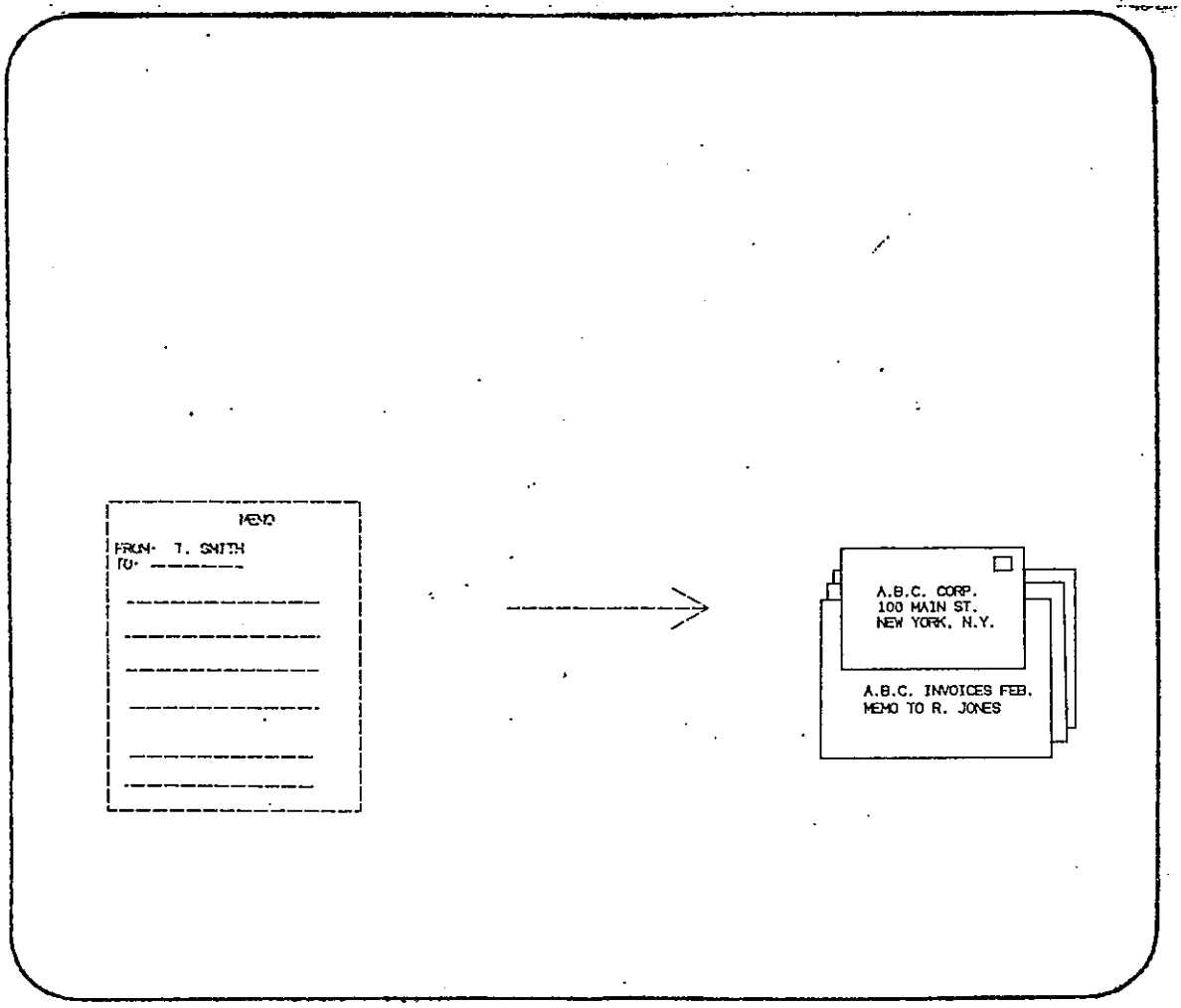

Figure 13 "Undo" anticipation

anticipate some of his needs for document retrieval, and to explain to the user why it has done what it has done whenever he chooses to ask. AI augmentation is discussed at length in a companion report (Kolodner).

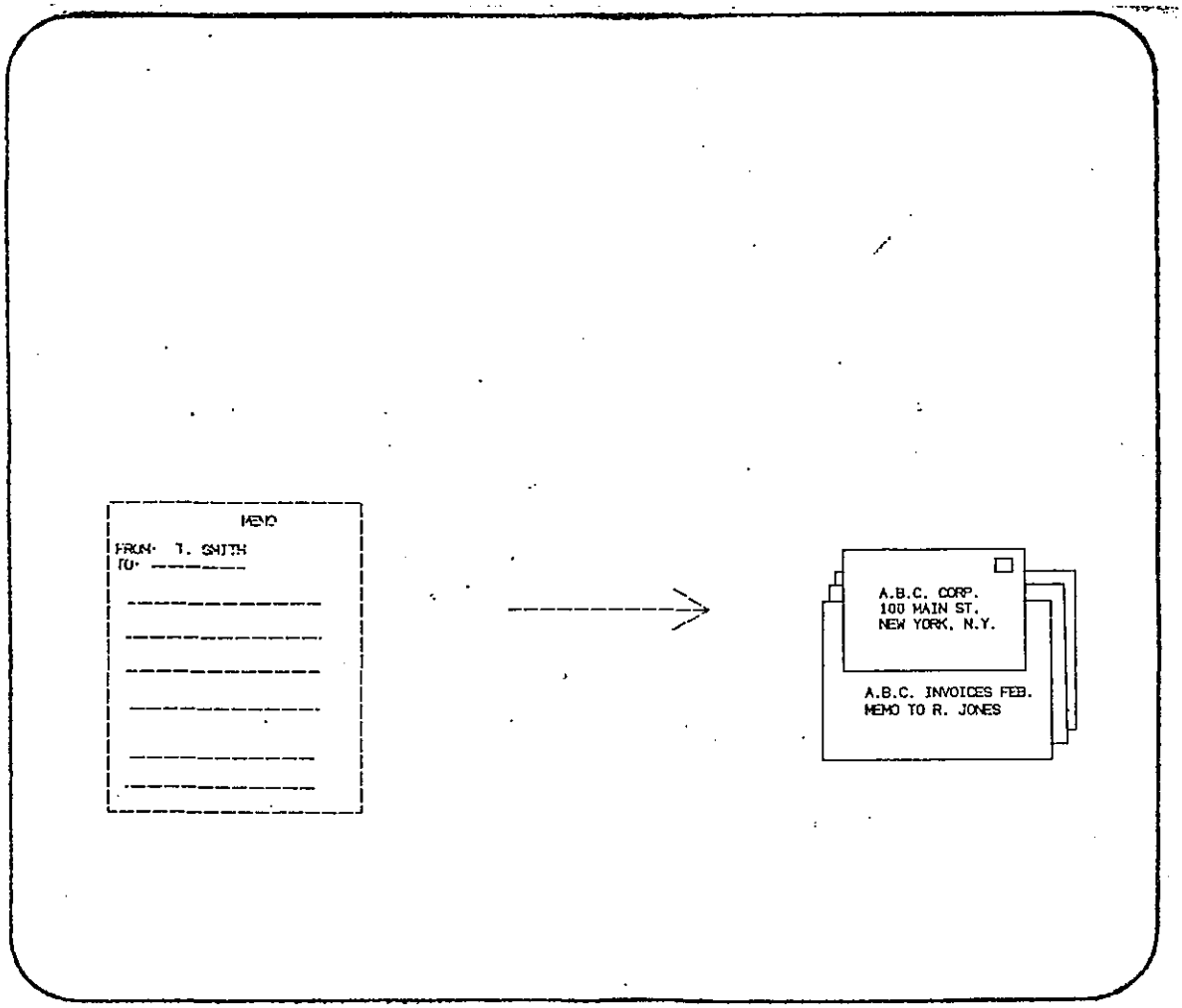

Figure 13 "Undo" anticipation

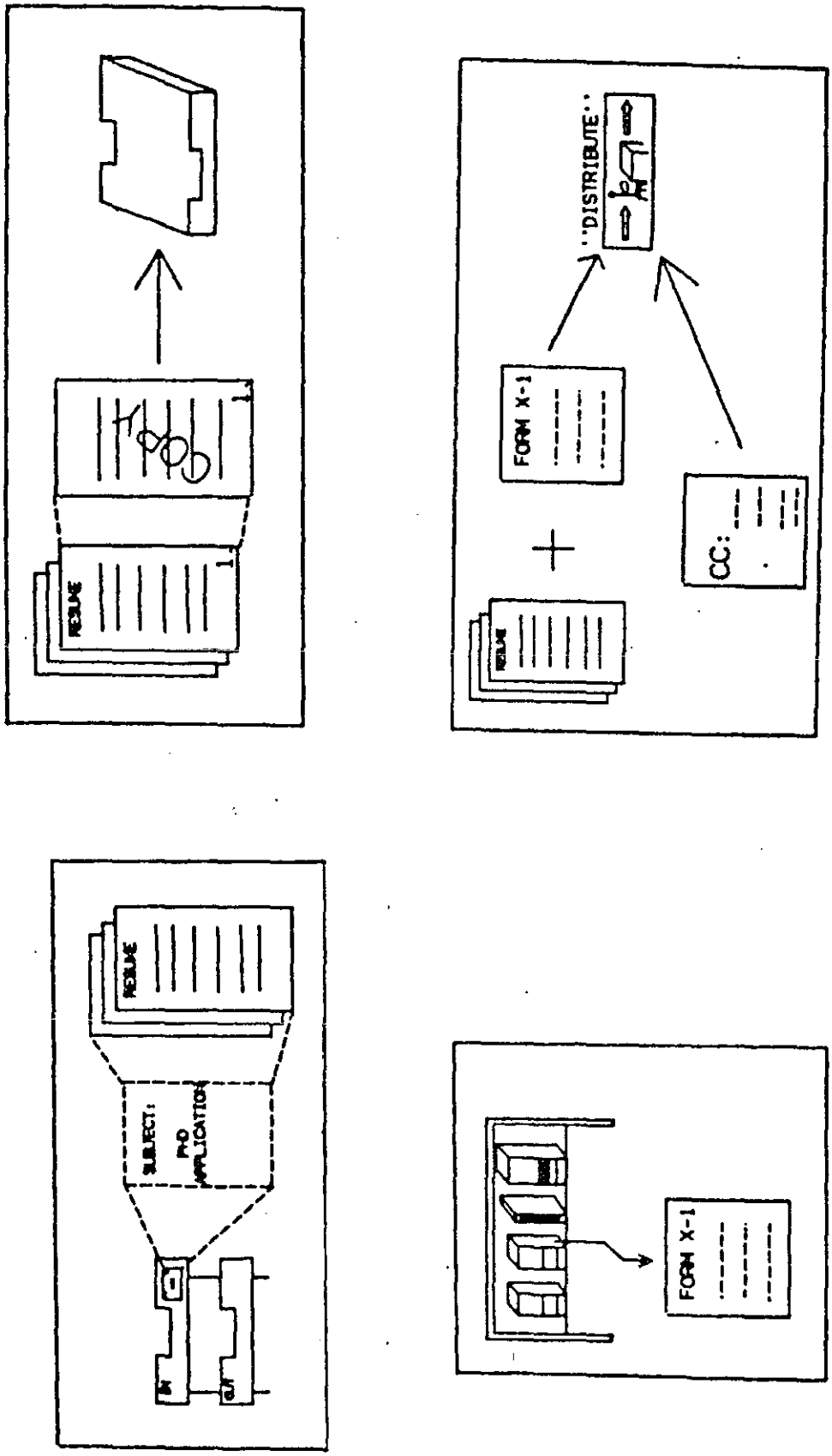

Figure 14 Procedure icons

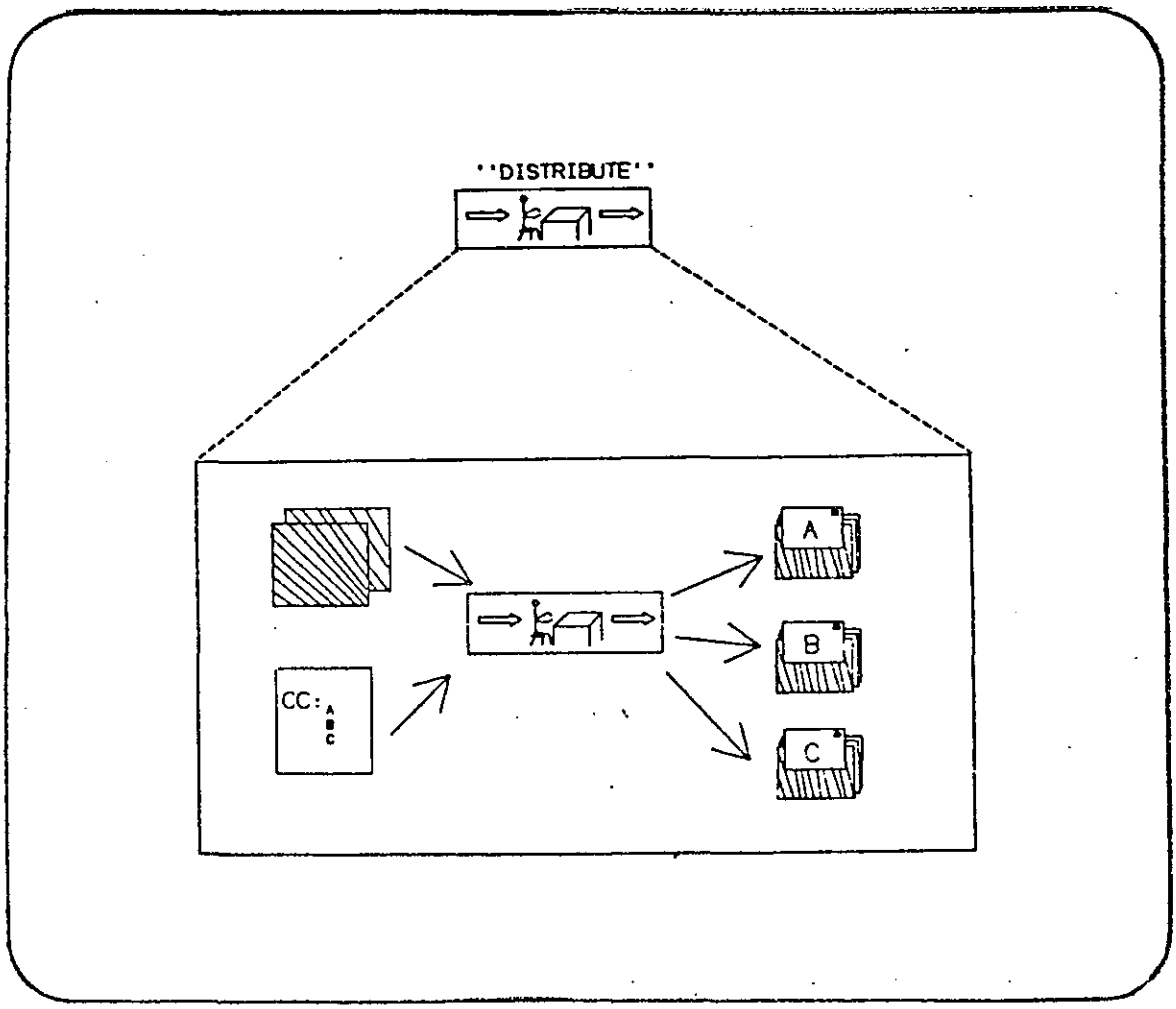

Figure 15 Distribution subprocedure

8. Implementation of Pictureworld Objects

In this section we outline a software architecture for implementing a Pictureworld system. The design strategy is to simulate a set of abstract office objects. In doing this, some use is made of the notions of data abstraction.

Pictureworld objects are visual and conceptual entities that a user manipulates on the display screen. They are the objects discussed in previous the sections. The user never has to deal with any other notions. The term *interface object* refers to the internal implementation of a Pictureworld object. All functions available to the user are associated with one or more of these objects, and he is unaware of support functions on which they are built. The implementation of a particular interface object (e.g., file cabinet) will normally make use of various system support facilities such as "disk file manager" and "storage manager." These *implementation objects* will never be manifested to the user directly. Thus we recognize two classes of data abstractions, interface objects and implementation objects. The general form of interface objects is presented here in some detail. Implementation objects provide the typical sort of general purpose support, found in all operating systems, for file and storage management, communications, and program execution. They are not discussed here.

Interface Objects:

Each interface object has five primary components:

* interactive input controller,
* display generator,
* operation set and message handler,
* encapsulated data, and
* contained objects.

Interactive input controller: Some user interactions require immediate, reflex-like visual feedback. Touching an object on the screen causes the object to brighten to confirm its selection; or, pressing a key causes the cursor to jump to the next "input field." Such responses are relatively simple (e.g., they require no file management), and make only minor modifications to the display (e.g., highlighting, cursor movement, echoing of keyed input). These functions are normally handled by a keyboard/display controller. Such controllers may be programmable to the extent that display fields have function attributes like "writable", "highlighted", "lightpen detectable," but two problems arise: (a) We wish to support a richer variety functions (e.g., highlighting when touched or when cursor passes over some boundary); (b) We want to be able to have many different objects displayed on the screen

simultaneously and have each tailored to respond, visually, in its own way. Modular design requires that each object defines its own interactive input controller. An interactive input controller is, therefore, a small program associated with a portion of the display screen and possibly with other I/O gear (e.g., keyboard, touch panel). The portion of the display occupied by an interface object will become a virtual display for that object. When the virtual display is referenced (e.g., by cursor, touch), the corresponding interactive input control program will get control. In principle, these programs should be able to be down-loaded on-the-fly from a host processor into the real I/O controller where they will be ready to execute when needed. In addition to handling immediate display feedback, they function as drivers for their more complex counterparts in the host processor. They communicate with the host via a message protocol. For example, the file cabinet search query may be completely formulated on the screen under prompting control of the file cabinet interactive input controller. When the query is completed it is sent to the host in the form of a message where it is ultimately processed by a file manager. The message protocol gives us a high degree of flexibility. For example, this protocol makes it easy to allow the user to interact with the calendar while a file cabinet search is being processed.

There are two kinds of statuses associated with any interface object -- functional status and visual status. Functional status refers to the logical and informational state of an object and its intrinsic data -- for example: whether or not the desk currently has a letter on it; or whether or not the file cabinet is currently being searched, and if so for what? Visual status refers to how the object currently appears on the display screen. We envisage that each object may have as many as three visual forms, each displaying different amounts of detail about the current functional status of the object. The two statuses are maintained independently. The detailed functional status of an object is not lost when it is being displayed in a reduced, summary form.

Display generator: Each object has a display generator component. It is a program which resides in the host processor and has two functions. Depending on the current visual and functional statuses of the object, it composes the actual display image (in an appropriate graphic representation) for the object. It also constructs the interactive input control program associated with that display image. The display images and their interactive input control programs are collected together and loaded into the real I/O controller. It is anticipated that the output of the display generator programs will be mostly tables which will be interpreted by a program in the I/O controller.

Together, the display generator and the interactive input control programs for each interface object manage the display and associated workstation I/O gear. This architecture, along with the message protocol, is designed to allow the workstation to be programmed to display and manipulate different objects in an interleaved fashion with a minimum of inter-object software dependencies. Those objects that have more than one visual status will be required to support operations that accomplish status changes. For example, the file cabinet will support an "enlarge" operation that will make it the focus of attention on the screen. It will be implemented by having its interactive input control program recognize a reference to the small file cabinet (e.g., touch panel or cursor select) and have it send a "enlarge file cabinet" message to the host. This message will be filtered through the office object which will pass it along to the file cabinet object after sending a "shrink" message to the object that is currently "enlarged," if any. More on this process below.

Operation set: We see there are two classes of operations, those that affect only visual status (as discussed above), and those that affect functional (as well as visual) status. It is this later class that represents the set of functions available to the user. Functional operations normally affect the data associated with an object (e.g., a file search creates a search response document), or they can merely send messages to other objects (e.g., move a piece of mail from the in-box to the desk).

Encapsulated data: Each data item (e.g., calendar entry, document) is encapsulated within a particular object. It can be accessed only through that object, and only be means of the operations defined for that object. As discussed below, objects often contain other objects. For example, the in-box can contain envelopes. Containing objects normally restrict the set of operations currently allowed on their contained objects (e.g., you can view envelopes when they are in the in-box, but to view their contents they must be moved to the desk object.) Whereas data encapsulation is fixed, object containment is mutable and often transient.

Message handler: Objects always communicate via a message protocol. This protocol supports communications between separate processors (e.g., between the I/O controller and host), and also allows a high degree of multi-programming among individual objects. In fact, the proposed architecture models each object as an autonomous process containing its associated data; the process exists as long as the object exists. When the object receives a message, it wakes up and performs the requested operation. When one object communicates with another object it sends it a message; a message can be examined by one object and passed to another unmodified or with changes. Those objects that

are primarily concerned with communicating between other objects (e.g., the arrow) implement most of their operation set through a message handler.

Contained objects: Finally, objects are organized into a containment tree. The office object contains all the major interface objects (e.g., in-out box, desk, file cabinet). Each of these in turn contains other objects (e.g., envelopes, documents), and so forth down to the smallest objects in the system. Although in some cases it would be desirable to address words, or even characters, as objects, this level of granularity may be too small to be practical.

Given that each interface object has the five components outlined above, we can now describe the sequence of events surrounding a single user interaction with the system. A message to the host processor is first routed to the office object. The office object keeps track of the visual status of each object with respect to its current size and position. It insures that there is no overlapping of virtual displays, and in effect keeps track of the current focus of attention. The office object then passes the message to the appropriate interface object where the specified operation is executed. In general, both the functional and visual status of an object may change as a result of applying one of its operations. The office object is notified when the visual status of an object changes, and this initiates a new display generation sequence. The display generator for each object that has changed is called, and they may in turn call the display generators of included objects. When a display generator is called, the size and position of the icon is specified by the containing object. Each display generator decides which version of its icon to draw, based partly on the specified size. The newly created display generator outputs are collected by the office object and down-loaded into the I/O controller, ready for the next sequence of user interactions.

9. Conclusion

Several important gaps in our present knowledge suggest the need for careful research in systems for the support of office principals. Areas of ignorance include: the effects of system design parameters on user acceptance, effects on productivity gains, effects on office sociology, methods for evaluating and justifying systems, and numerous cost-effectiveness tradeoffs.

While we cannot yet define and implement cost-effective systems, enough is known to design, implement, and test some experimental systems. We have described a system concept, Pictureworld, which has many of the properties which we believe to be important to principal support, as a candidate system for inclusion in a comprehensive program of research in this field.

In particular, the iconic, natural, self-prompting user interface of Pictureworld gives the user very low entry cost. In addition, we believe that it minimizes cognitive interference between the secondary task of commanding the system and the user's primary task, which the system is supporting. The functions we have defined are believed to be sufficient for an entry level system. The organization of the software into Pictureworld objects is expected to facilitate expansion and maintenance of the system.

While the high quality display needed for Pictureworld may pose a cost problem, it seems premature to dwell on that point. We are presently in need of establishing value.

10. References

Kintsch, Walter (1977), Memory and Cognition, New York, John Wiley and Sons, (pp 239-244).

Kolodner, Janet, Design for an Intelligent Office System. Research Report RC8385. IBM Thomas J. Watson Research Center.