Object Lens: A "Spreadsheet" for Cooperative Work

Kum-Yew Lai

Thomas W. Malone

Keh-Chiang Yu

90s: 89-071

Sloan 2053

September 1988

This paper will appear in *ACM Transactions on Office Information Systems*, in press

Management in the 1990s
Sloan School of Management
Massachusetts Institute of Technology

Acknowledgements

The work described in this paper was supported, in part, by Wang Laboratories, Xerox Corporation, General Motors/Electronic Data Systems, Bankers Trust Company, the Development Bank of Singapore, and the Management in the 1990s Research Program at the Sloan School of Management, MIT.

We would especially like to thank Ken Grant who suggested some of the earliest ideas that led to Object Lens and Jin Lee who helped debug the most recent version. The Object Lens system and this paper have also benefitted from conversations with Cheryl Clark, Kevin Crowston, Randy Davis, Frank Halasz, Mitch Kapor, Stan Lanning, Wendy Mackay, Ramana Rao, Randy Trigg, David Rosenblitt, and Franklyn Turbak.

Abstract

Object Lens allows unsophisticated computer users to create their own cooperative work applications using a set of simple, but powerful, building blocks. By defining and modifying templates for various *semistructured objects*, users can represent information about people, tasks, products, messages, and many other kinds of information in a form that can be processed intelligently by both people and their computers. By collecting these objects in *customizable folders*, users can create their own displays that summarize selected information from the objects in table or tree formats. Finally, by creating *semiautonomous agents*, users can specify rules for automatically processing this information in different ways at different times.

The combination of these primitives provides a single consistent interface that integrates facilities for object-oriented databases, hypertext, electronic messaging, and rule-based intelligent agents. To illustrate the power of this combined approach, we describe several simple examples of applications (such as task tracking, intelligent message routing, and database retrieval) that we have developed in this framework.

1. INTRODUCTION

It is common in the computer industry today to talk about the "next spreadsheet"--to claim that a particular application will be the "next spreadsheet" or to wonder what the "next spreadsheet" will be (e.g., [Greif, 1988]). Usually the term "spreadsheet" is used in this context simply to connote a product that embodies some kind of design breakthrough and is very successful.

We will focus here on a more specific property of spreadsheet programs: They make a restricted, but nevertheless very flexible and useful, set of computational capabilities extremely easy to use. It is, of course, possible to do any computation a spreadsheet can do in a general purpose programming language. But because doing these things with a spreadsheet program is so much more convenient, the number of people who can use computers to do them increases by orders of magnitude.

In this paper, we will describe an early prototype of a system, called Object Lens, that we believe shares this property of spreadsheets: It makes accessible to unsophisticated computer users a set of computational and communications capabilities that, while limited, are quite flexible and useful for supporting a wide variety of cooperative work activities. In other words, we use the term "spreadsheet" here, not to connote financial modeling or constraint languages, but to connote a flexible infrastructure in which people who are not professional programmers can create or modify their own computer applications.

In the remainder of this paper, we will (1) describe the key ideas used in the design of Object Lens, (2) show how these ideas are realized in Object Lens features, and (3) illustrate the flexibility and usefulness of these ideas in several examples of cooperative work.

1.1 Three views of Object Lens

Before proceeding, it is useful to point out three ways of viewing the Object Lens system:

(1) *Object Lens is the "second generation" of the Information Lens system.* Object Lens is based on our experience with using and enhancing the Information Lens (Malone, Grant, Turbak, Brobst & Cohen, 1987; Malone, Grant, Lai, Rao, & Rosenblitt, 1987), an intelligent system for information sharing and coordination. A very large number of the enhancements that we and others have suggested for the Information Lens are included in Object Lens. Like the Information Lens, Object Lens uses ideas from artificial intelligence and user interface design to represent knowledge in such a way that both people and their computational agents can process it intelligently. Object Lens, however, is a significant generalization of the Information Lens. It potentially goes far beyond the Information Lens in the kinds of knowledge that can be represented and the ways that information can be manipulated.

(2) *Object Lens is a user interface that integrates hypertext, object-oriented databases, electronic messaging, and rule-based intelligent agents.* Object Lens does not include all the capabilities of all these different classes of systems, but we have been surprised at how cleanly a large portion of these diverse capabilities can be integrated. The key contribution of Object Lens is thus not the completeness of its implementation, but the integration of its user interface. Since the capabilities of these different kinds of systems are no longer separate applications, each capability is more useful than it would be alone, and the resulting system is unusually flexible.

(3) *Object Lens is a knowledge-based environment for developing cooperative work applications.* In the original Information Lens system, we developed specific applications for information sharing, meeting scheduling, project management, and computer conferencing. From the viewpoint of knowledge-based systems, these applications only included knowledge about different types of messages: the kinds of information the messages contained and the kinds of actions they could evoke. Object Lens, in contrast, can include explicit knowledge about many other kinds of objects, such as people, tasks, meetings, products, and companies. We expect that the flexible tools Object Lens provides for dealing with these diverse kinds of knowledge will significantly increase the ease of developing a much wider range of applications. This last view of Object Lens, which emphasizes its flexibility, is our primary focus in this paper.

2. KEY IDEAS

One of the most important characteristics of Object Lens is that it is a *semiformal system*. We define a semiformal system as a computer system that has the following three properties: (1) it represents and automatically processes certain information in formally specified ways; (2) it represents and makes it easy for humans to process the same or other information in ways that are not formally specified; and (3) it allows the boundary between formal processing by computers and informal processing by people to be easily changed.

Semiformal systems are most useful when we understand enough to formalize in a computer system some, but not all, of the knowledge relevant to acting in a given situation. Such systems are often useful in supporting individual work, and we believe they are especially important in supporting cooperative work where there are usually some well-understood patterns in people's behavior, but where there is usually also a very large amount of other knowledge that is potentially relevant but difficult to specify.

In order to create such a flexible semiformal system, the knowledge embodied in the system must be exposed to users in a way that is both *visible* and *changeable* (cf., Turbak, 1986). That is, users must be able to easily see and change the information and the processing rules included in the system. In Object Lens, there are three key ideas about how to represent and expose knowledge to users:

(1) "Passive" information is represented in *semistructured objects* with template-based interfaces;

(2) "Aggregate" information from collections of objects is summarized in *customizable folders*; and

(2) "Active" rules for processing information are represented in *semiautonomous agents.*

In the remainder of section 2, we will provide an overview of how these two components allow us to expose knowledge to users in a way that is both visible and changeable. Detailed descriptions of the system features are in section 3.

2.1 Semistructured objects

Users of the Object Lens system can create, modify, retrieve, and display objects that represent many physically or conceptually familiar things such as messages, people, meetings, tasks, manufactured parts, and software bugs. The system provides an interface to an object-oriented database in the sense that (1) each object includes a collection of fields and field values, (2) each object type has a set of actions that can be performed upon it, and (3) the objects are arranged in a hierarchy of increasingly specialized types with each object type "inheriting" fields, actions, and other properties from its "parents" (see Dittrich & Dayal, 1986; Shriver & Wegner, 1987; Stefik & Bobrow, 1986). For example, a TASK object may have fields like Requestor, Performer, Description, and Deadline; a PERSON object may have fields like Name, Phone, Address, and Job title; and a STUDENT object may add fields like Year and Advisor to the fields present in all PERSON objects. Some objects (e.g., MESSAGES) have specialized actions defined for them (e.g., Answer and Forward). As described in more detail below, we have provided rudimentary facilities for saving and sharing objects, and we are currently exploring ways to link our interface to remote databases.

The objects in Object Lens, like messages in the Information Lens, are *semistructured* in the sense that users can fill in as much or as little information in different fields as they desire and the information in a field is not necessarily of any specific type (e.g., it may be free text, such as "I don't know").

2.1.1 *Template-based user interfaces.* Users can see and change objects through a particularly natural form of template-based user interface. These interfaces have a number of virtues. For instance: (1) they resemble forms, with which users are already familiar, (2) they conveniently inform users about the fields contained in an object and about other information such as the likely alternatives for different fields, and (3) their use is consistent across many different kinds of objects.

We will see below how this interface approach, which was used for messages and rules in the Information Lens, can be easily generalized to many different kinds of objects.

2.1.2 *Relationships among objects.* Users can easily see and change the relationships among objects by inserting and deleting *links* between the objects. For instance, the Requestor and Performer fields of a Task object might contain links to the Person objects that represent, respectively, the person who requested that the task be done and the person who will perform the task. Then, for instance, when the user looks at the Task object, it will be easy to get more information (e.g., the phone numbers) about the people involved with the task. We will see below how this capability of linking objects to each other provides a rudimentary *hypertext* system as a special case (see Conklin, 1987, for an extensive review of hypertext systems). We will see below how it is also possible for an object to which a link appears to be displayed as an *embedded template* inside the original template.

2.1.3 *Tailorable display formats.* Users have several options for changing the ways they see objects. For instance, they can easily: (1) select which fields will be shown and which will be suppressed, (2) rename selected fields, and (3) specify the default and alternative values the system presents for individual fields.

2.1.4 *Inheritance hierarchy for objects.* The creation and modification of type definitions is simplified by arranging object types in an inheritance hierarchy (e.g., Stefik & Bobrow, 1986). New types of objects are defined as specializations of existing object types, and they automatically "inherit" all properties of the existing objects except those which are specifically "overridden." Since most of the information about new object types can thus be "inherited" from existing types, rather than having to be re-entered each time, creating new object types becomes simpler. Also, when an object type definition is changed later, the changes are automatically "inherited" by the specializations of that object type.

2.2 Customizable folders

Users of Object Lens can group collections of objects together into special kinds of objects called Folders. For instance, folders can be created for groups of people (e.g., project teams, company directory), tasks (e.g., those completed, those to be done by you, those to be done by others), messages (grouped according to topic or urgency), and so forth. Users can also easily customize their own displays to summarize the contents of objects in a folder. For instance, they can select certain fields to be displayed in a *table* with each row representing an object in the folder and each column representing a field. They can also select fields from which the links between objects will be used to create a *tree* (or graph) display with each object represented as a node in the tree and each link in the selected field represented as a line between nodes.

2.3 Semiautonomous agents

Users of the Object Lens system can create rule-based "agents" that process information automatically on behalf of their users (see Crowston & Malone, 1988, for an extended discussion of agents). These agents provide a natural way of partitioning the tasks performed automatically by the system. As we will see below, agents can be "triggered" by events such as the arrival of new mail, the appearance of a new object in a specified folder, the arrival of a pre-specified time, or an explicit selection by the user. When an agent is triggered it applies a set of rules to a specified collection of objects. If an object satisfies the criteria specified in a rule, the rule performs some prespecified action. These actions can be general actions such as retrieving, classifying, mailing, and deleting objects or object-specific actions such as loading files or adding events to a calendar.

The agents in Object Lens are "autonomous" in the sense that once they have been created, they can take actions without the explicit attention of a human user. They are only "semiautonomous," however, in the sense that (a) they are always controlled by a human user (that is, all their rules can be easily seen and changed by their human user), and (b) they may often "refer" objects to their human user for action (e.g., by leaving the object in the user's inbox) rather than taking any actions on their own.

2.3.1 *Descriptions*. Since agents and rules are themselves objects, users can see and modify them with the same template-based user interface that is used for all other kinds of objects. To specify the criteria for when a rule should act upon a given object, users create *descriptions* of the objects to which the rules apply. A description is simply a partially filled-in template for an object of a particular type. Descriptions can also include *embedded descriptions* that specify characteristics that must be satisfied by objects to which the original object is linked. For instance, a description of a Task might include an embedded description of the Person who will perform the task. These embedded descriptions (like those in the Rabbit system [Tou et al, 1982]), allow users to easily specify object retrieval operations that are equivalent to "joins" followed by "selects" in a relational database.

3. SYSTEM FEATURES

In this section, we will describe in more detail the basic system features of Object Lens and illustrate them with simple examples (see Lai [1987] for more details about an earlier version of the system). The Object Lens system is implemented in Interlisp-D on Xerox 1100 series workstations connected by an Ethernet. The system makes heavy use of the object-oriented programming environment provided by Loops and the built-in text editor, Tedit. Except where otherwise noted, everything described here has been implemented, but many features have not yet been extensively tested. As of this writing, the basic mail handling capabilities have been used regularly by two people in our development group for about 6 months and the other facilities have received limited testing.

3.0.1 *Terminology: Objects and templates.* Before proceeding it is helpful to clarify some terminology concerning objects and templates. First, we distinguish between *object types* (or "classes") and specific *object instances* (e.g., see Fikes & Kehler, 1985). We use the term *object type* to refer to a kind of object (such as Person or Task) and the term *object instance* (or simply "instance") to refer to a specific example of one of these object types (e.g., "Joe Smith or "Task No. 17"). In contexts where the distinction between object types and object instances is not critical, we use the term *objects* to include both.

We also use the term *template* in two ways. First, in a general sense, we use the term *template* to mean any semi-structured collection of fields and field contents. Most of a user's interactions with Object Lens are based on such templates. Second, in the Object Lens screen displays, we use the word Template to mean object type definition. (When we use Template in this specialized sense, we will always capitalize it.) For instance, users can change the display format for all Person objects by editing the Template that defines the Person object type.

3.1 Editing instances

Figure 1 shows a template for an instance of a Person. Using the built-in text editor, users can insert text or bitmaps in any field. In addition, when users click on a field name with the mouse, a list of likely alternative values for that field appears in a pop-up menu. The alternatives may be links to other objects or just text strings. Selecting one of these alternatives causes the alternative to be automatically inserted in the field. For instance, the figure contains a link to the Person object representing Kum-Yew Lai's supervisor. To insert links to objects that are not in the alternatives list, the user (a) positions the cursor at the place in the template where the link is to be inserted, (b) selects the Add Link option from the menu at the top of the window, and then (c) points to the object to which the link should be made. After a link is inserted, clicking on it with the mouse causes the object it points to to appear on the screen.

In the current version of Object Lens, users can insert any combination of text, numbers, links, and bitmaps in any field. Then, in some cases, type checking is done when the editing window for the instance is closed or when certain kinds of processing are done. For instance, the To and cc fields are checked for valid addresses before sending messages and the "move to" field in rule actions is checked for valid folders (see below for descriptions of rules and folders). In future versions of Object Lens, we may experiment with more restrictive type enforcement in certain fields. For instance, it should probably be impossible to even insert something other than a folder in the "move to" field of a rule action.

Figure 2 shows a slightly more complex template; this one is for a Bug Fix Request message. One of the fields of this template is the Bug to be fixed and the value of this field is a link to a Bug object. In this case, instead of simply showing a link to the Bug object, the template contains an *embedded template* for the Bug object itself. The fields in this embedded template can be edited just like the rest of the fields in the template. We will see below how users can specify whether links to other objects should be displayed as *link icons* (as in Figure 1) or as *embedded templates* (as in Figure 2).

3.2 Creating new instances

To create and display a new instance of an object type that already exists, users click with the mouse on the definition (i.e., the Template) for that object type. Figure 3 shows the Templates currently included in our system. For instance, to send a new message, users click on the Template for the type of message they want to create; to create a new person object, users click on the Person Template. Then an object instance, like those shown in Figures 1 and 2, will appear and the user can fill it in.

3.3 Creating new object types

To create a new object type, users click (with the middle mouse button, instead of the left one) on the Template for the "parent" object type (see Figure 3). This causes a menu to appear showing alternative actions that can be performed on a Template. One of these actions is to Create a subtemplate. When the user selects this action, a new Template is created with all the fields and properties of its "parent." Then users can add fields to the new Template or change its display format and other properties (see below).

In the current version of Object Lens, all Things have three fields: Name, Keywords, and Comments. All objects inherit these fields, though as discussed below, some objects rename these fields or suppress their display. For instance, Messages rename the Name field to be Subject and the Comments field to be Text.

3.4 Changing the display format and other properties of object types

To change the display format or other properties of an object type, users "edit" the Template that defines the object type. Users make these changes by selecting actions from the menu that appears when they click on the Template (as shown in Figure 3) with both mouse buttons. In this way, users can change (a) which fields of the object are actually displayed, (b) the names of the fields that are displayed, (c) the alternative values that are displayed for each field, (d) the default values that are displayed in each field when new instances are created, and (e) whether the links in a field should be shown as link icons (see Figure 1) or as embedded templates (see Figure 2). In this mode, users can also add or delete fields from a template. All the changes made to a template are applied to old instances of an object type as well as to newly created ones. For example, if the user changes the name of a field, then the new name will be shown when any old instances are redisplayed.

We anticipate that this system will be used with a core set of object types shared by the users in a group and that the fields in these types will be modified only by an "authorized view administrator." Other users will be able to change the display format of these types (e.g., suppress the display of a field or change its name), but they would not be able to delete or add fields to these "official" types. All users would, however, be able to create their own types as specializations of the official types, and for these types they could add and delete new fields as desired. Elsewhere (Lee & Malone, 1988a, 1988b) we have proposed a scheme for letting an arbitrarily large number of groups share partially overlapping sets of type definitions in arbitrary ways. One of the key ideas of this scheme is that specialized types created by one group can be interpreted by members of another group as instances of the most specific "ancestor" type that both groups share. For instance, a "Student" object created by one group might be interpreted as a "Person" object by another group that does not have a definition for "Student."

3.5 Folders

As noted above, Object Lens users can group collections of objects together into special kinds of objects called Folders (see Figure 7). An object can be added to a folder in two ways: (1) automatically, as the result of a rule action, or (2) manually using the Add Link action from the *Others* submenu on the folder. In both cases, the folders will contain links to the objects, not the objects themselves. Therefore, the same object can appear in more than one folder. Other actions for moving, copying, and deleting both objects and links are described below.

Object Lens currently provides two formats for displaying the contents of folders: *tables* and *trees*. Tables show the values of selected fields from the objects contained in the folder. For instance, Figure 7a shows a folder that contains objects representing people with the fields displayed for a simple office directory. Users can easily tailor the format of these displays by selecting from a menu the fields they want to have included in the table. For instance, Figure 7b shows the same folder, but with the display format changed to include a different set of fields.

Trees are graphs that show the objects in a folder and the links that connect these objects. Just as users can select the fields to be shown in a table, they can also select the fields from which links will be shown. For instance, Figure 7c shows the same folder again, but this time in tree format with the "Supervisor" field selected as the one from which links are displayed. In this case, the display resembles a simple organization chart. In the current version of Object Lens, only the links in one field at a time can be displayed in a tree. In future versions, we plan to allow links from multiple fields to be shown with the links from different fields being displayed as different types of lines (e.g., solid, dotted, etc.).

When a new folder is created, the user is asked to select the default object type to be contained in the folder. The user is then allowed to choose from the fields of this default object type when selecting the fields to show in a table or when selecting the fields from which links will be shown in a tree. Even though all folders have default object types, no strict type checking is enforced. If an object of an unexpected type is inserted into a folder, only the fields it shares with the default type are displayed in tables and trees.

3.6 Performing actions on objects

In addition to editing the contents of objects, users can also perform pre-defined actions on them. The actions that can be performed at any time depend on two primary factors: (1) the type of object being acted upon, and (2) the context in which the action is invoked.

3.6.1 *Object specific actions.* Each object type has a set of actions that can be performed on it. Some of these actions are "inherited" directly from the "parents" of the object type. Others may be modified or added specifically for this object type. For instance, there are some actions, such as Hardcopy and Save that can be performed on all objects (i.e., all instances of Thing and all its subtypes). (Some of these actions, such as Hardcopy, are not yet implemented for all object types.) In addition, more specialized types of objects have other actions defined for them. For instance, agents have a Run action that triggers them to start running, and folders have a Change Display Format action that changes them from table format to tree format or vice versa.

In a few cases, the object specific actions depend, not just on the type of the object, but also on its state. For instance, messages created on the local workstation have a Send action, and messages received from elsewhere have actions such as Answer and Forward. So far these state-specific actions on objects are implemented as special cases. However, we would like to experiment with a more general mechanism for representing state-specific actions and perhaps making this representation accessible to users. In some ways, this mechanism would be a generalization of the conversation manager in the Coordinator (Winograd, 1988) which restricts the types of messages that a user can send at a given point in a conversation, based on the conversation state.

3.6.2 *Context specific actions.* There are some actions that can be applied to any kind of object, but which can be invoked only from certain contexts. The primary contexts are: (1) from an editor (like the one in Figure 1), (2) from a folder that contains the object, (3) from a rule operating on the object, and (4) from a link icon for the object.

For instance, when an object is being displayed in an editor, there are several kinds of actions, such as Close, Move, and Shape, that apply to the editing window. Other actions in an editor include: (a) Add Link, (insert at the current cursor position a link to another object selected by the user), and (b) Cancel (close the window without saving any of the changes made since the window was last opened).

When an object is displayed in a folder, other context-specific actions can be applied to it, such as: (a) Show (open an editor on the object), and (b) Select (select the item for some later folder action such as Delete Selection).

The actions that can be applied to an object by rules are discussed in Section 3.7 below. The actions that can be applied to link icons include: Show (open an editor on the object), and Delete (delete this link to the object).

3.6.3 *Displaying and invoking actions*. Users invoke the above actions in slightly different ways depending on the context in which the object is displayed. If the object is displayed in an editor (like the one in Figure 1), then several of its most common actions are shown across the top of the editor, and all the other actions are shown in a menu that pops up when the *Others* action is selected.

When a link to an object is displayed (either as a link icon or as a row in a table), users can invoke actions in two ways. First, if users click on the link with the middle mouse button, a menu pops up showing all possible actions on the object. In addition, simply clicking on the link with the left mouse button invokes the most common action. For instance, clicking with the left button on a row in a table Selects the object for subsequent folder actions, while clicking with the left button on a link icon inside an editor Shows the object in another window on the screen.

3.7 Creating agents and rules

In some cases, agents can take actions automatically on behalf of their users. For instance, Figure 4 shows an example of a simple agent designed to help a user process incoming mail. When an agent is triggered, it applies a set of rules to a collection of objects in a folder. The agent in Figure 4 is applied to objects in the New Mail folder and is triggered by the arrival of new mail. That is, when mail is retrieved to the workstation, the mail program automatically inserts links to the new messages into the user's New Mail folder and these New Links trigger the agent. In the current version of Object Lens, two other kinds of automatic triggers are available: Daily at Midnight, and On the Hour.

The agent shown in Figure 4 includes several rules, one of which is shown in Figure 5. A rule contains an "IF" field (predicate) and a "THEN" field (action). Both these parts of the rule contain links to other objects which are shown as embedded templates. The IF part of the rule is a *description*, a special kind of template that describes a set of instances in terms of the values of their fields. The THEN part of the rule is an Action object.

To construct the IF part of a rule, a user (a) clicks on the IF field with the middle mouse button, (b) selects "Descriptions" from the menu presented, and then (c) selects an object type from the tree of object types presented. This causes a description of the appropriate type to be inserted in the rule as an embedded template, and the user can then fill in the fields in this description to specify the values that must appear in particular fields for an object to satisfy the rule. As in the Information Lens, more complex specifications for a field can be constructed by combining strings with *and, or, not,* and parentheses (i.e., arbitrary Boolean combinations are possible within a field). If specifications appear in more than one field, then all specifications must be satisfied at once for the rule to succeed (i.e., specifications in different fields are implicitly *and*-ed). As in the other template-based editors in Object Lens, pop-up menus listing likely alternatives for a field are available in editing descriptions.

To specify the THEN part of a rule, a user simply clicks on the THEN field and selects an action from the menu of alternatives presented. These actions are applied to the "current object" (the object matched by the IF part of the rule) in the context of the "current folder" (the folder specified in the "Apply to" field of the agent). In some cases (such as the "Move" action shown here), the user also needs to fill in some fields in the embedded template for the action (e.g., the field specifying where the object is to be moved). The actions currently implemented in rules include the following: "copy" (add the current object to a different folder without removing it from the current folder), "move" (add the current object to a different folder and delete it from the current folder), "delete" (remove the object from the current folder), and "add keyword" (add the specified keyword to the Keywords field of the object). In addition, rules can invoke object specific actions, including the actions that apply to all objects such as "hardcopy" and "save". We view the addition of more rule actions (and possibly the refinement of the rule syntax) as one of the important directions for our ongoing research.

The rules are applied in the order in which they appear in the agent's rule folder. Users can create extended reasoning chains by having some rules set characteristics of objects (using the Add Keyword action) which other rules test (by checking the Keyword field).

3.7.1 *Embedded descriptions.* With the capabilities we have described so far, all rules must depend only on information contained in the objects to which they are being applied. For instance, a rule about a message can depend only on information contained in the message itself. It is often desirable, however, to be able to specify rules that also depend on other information contained elsewhere in the knowledge base. For instance, in the Information Lens system, if a user wanted to specify a rule that applied to all messages from vice presidents, the rule would have to include in the From field, the names of all the vice presidents.

In Object Lens, it is possible to draw upon other information by having descriptions embedded within other descriptions. For instance, the rule shown in Figure 6 will be satisfied if the message is from any person with a job title that includes "vice president". To apply this rule, the system checks to see whether the string in the From field of the message is the same as the Name of any Person object in the knowledge base that satisfies the description.

3.8 Navigating through the system

The starting point for navigation through the Object Lens system is the Object Lens Icon, a window that shows whether the user has new mail waiting and includes a menu item to Show Basics (show the basic folders included in the system). The system folders accessible through the Show Basics action include: (1) a folder containing all the other folders in the system, (2) a folder containing all the

Templates defined in the system (Figure 3), (3) a folder containing all the agents defined in the system, (4) a folder for each object type containing all the instances of that type in the system, and (5) the New Mail folder, into which new mail retrieved from the mail server is automatically inserted. In addition, we have designed but not fully implemented two other folders: (6) Everything, a virtual folder containing all objects in the system, and (7) Orphans, a virtual folder containing all objects to which no links exist.

These basic folders provide users with convenient starting points for locating any object in the system. In relatively small systems, users can browse through these folders directly. In larger systems, we expect that users will let their agents search through the system folders to find objects that meet certain criteria. It is also possible for (a) individual users to create their own customized "directory" folders that contain the folders and other objects they most often use, and (b) application developers to create folders containing the objects used in their application.

3.9 Saving and sharing knowledge

One of the important research directions we plan to pursue in the Object Lens system involves different ways for people to save and share the kinds of knowledge described above. For instance, we are currently experimenting with linking Object Lens to a remote database server that contains large shared relational databases. This work is still at an early stage, but it is clear that the usefulness of Object Lens will be significantly enhanced if it includes access to shared databases. In the current version of Object Lens, we have preliminary solutions to the problems of saving and sharing knowledge that meet some, but not all, of the needs people will have in this area.

3.9.1 *Saving knowledge.* Users can save an object (or a collection of objects in a folder) at any time by performing the Save action on the object (or the folder). This action uses the file package commands from the underlying Loops and Lisp systems to store the objects in permanent files in a form that can be reloaded at any time. There is also a "Save" action on the main Object Lens icon that saves all the instances in the workstation.

The potential disadvantages of this approach to saving knowledge are that (1) it requires explicit user actions to save objects to permanent storage and (2) it requires all knowledge used by the system to be loaded onto the local workstation. Sharing remote databases will, of course, help solve these problems, but we expect that systems like Object Lens can be of value even without shared databases. For example, many users are already accustomed to explicitly saving their work in applications such as word processing, and even this task can be simplified by creating agents to run periodically (e.g., every night) and do automatic backups of selected objects.

3.9.2 *Sharing knowledge by sending messages*. There are two ways users of Object Lens can share objects with each other: (1) by sending messages, and (2) by transferring files. In this subsection, we discuss sending messages; in the next, we discuss transferring files. When an Object Lens user sends a message, the message object is converted into text and sent via the existing mail system. Any connected electronic mail users can receive and read this textual message. When an Object Lens user receives the message, it is added as a new object in the receiver's knowledge base.

When a user sends a message containing an embedded object that is expanded (as in Figure 2), the embedded object is converted into (indented) text in the message in a form that (a) can be easily read by any receivers who are not using Object Lens and (b) is reconverted into another embedded object when it is received by Object Lens users. When a user sends a message containing embedded objects that are *not* expanded (e.g., that are shown only as link icons), the names of the objects are included in the message in place of the link icons, but these names are not resolved back into link icons at the receiver's end.

One intriguing research direction here involves how to communicate embedded objects in such a way that they can be resolved into pre-existing objects at the receiver's end. For example, if the sender's message contains a link to a person object, it would be nice for the receiver's system to be able to automatically resolve this link into the receiver's object representing the same person.

3.9.3 *Sharing knowledge by transfering files*. The second way for users to share objects is by transferring files. As described above, it is easy for users to store on a file server the current state of a set of objects. Other users can then load these files to create (or update) the objects in their own workstations. Saving and loading these files can often be done automatically. For example, we expect that a common way for users to keep current versions of shared information such as names, addresses, and job titles of people in their organization will be to have someone maintain the official version of this information and periodically distribute updates to other users in the organization. Distributing these updates could be done in several ways: (1) the "maintainer" could have automatic agents that periodically store the current versions on a file server and the other users could have automatic agents that periodically load the most recent versions, or (2) the maintainer could explicitly send out messages announcing the availability of files containing updated objects and the other users could have agents that automatically load the files announced in such messages (e.g., a rule might load all files specified in "Official file update" messages from the official maintainer).

One potential problem with this approach is that any changes the users have made to their local copies of objects (e.g., any notes they had added in the Comments field) will be lost when a new version of the object is loaded. To help solve this problem, we are currently investigating more specialized updating actions for agents to use. With this approach, the official maintainer will be able to distribute update messages that specify changes in particular fields of particular objects. Users can then set up agents that make these updates automatically under most conditions, but under certain conditions the user might be notified before the update is made (e.g., if the field about to be modified has previously been changed by the user). In some cases, the user might want to have the change made automatically but also be notified (e.g., if someone in the user's group is changing phone numbers).

4. OTHER APPLICATIONS

In this section, we will give more examples of how the above features can be combined to create a variety of cooperative work applications.

4.1 Task tracking

One frequently mentioned capability for cooperative work applications is the ability to keep track of the tasks people are supposed to do (e.g., Winograd & Flores, 1986; Sluizer & Cashman, 1984). For instance, such systems can help answer questions like: What tasks have other people requested me to do? Are any of these tasks overdue? What tasks have I requested other people to do for me?

It is a straightforward matter to support capabilities like this in Object Lens. For instance, the system already includes message types for action requests and commitments. Even in the Information Lens, it was possible to automatically sort these messages into folders according to who is to perform the task, which project it involves, and so forth. In the Information Lens, however, the summary display of a folder's contents shows only the standard message header fields: From, Date, and Subject. To see more about the tasks, individual messages have to be displayed, one at a time. In Object Lens, the messages within a folder can easily be summarized by displaying whatever fields the user chooses. For example, Figure 8 shows a table display of action request messages that includes the action deadline.

4.2 Intelligent message sorting: Engineering change notices

As we have described in more detail elsewhere (Malone et al, in press), an intriguing example of a cooperative work problem involves disseminating information about changes in product specifications (often called "engineering change notices") to the appropriate people in an organization. It was already possible in the Information Lens to sort engineering change notices according to the contents of fields such as Part Affected, Type of Change, and Severity. In Object Lens, it is possible to use additional knowledge to do even more intelligent sorting. For instance, Figure 9 shows a rule that uses a doubly embedded description to select all change notices that involve parts for which anyone reporting to a particular manager is responsible.

4.3 Database retrieval

There are clearly many cases in both individual and cooperative work when it is useful to be able to automatically retrieve from a database objects that satisfy certain conditions. Object Lens provides a simple way to perform database queries: Users can simply create agents that scan the objects in one folder and insert links to selected objects into another folder. The rules in the agents specify the criteria for selecting objects.

For instance, suppose you wanted to find all the technical staff members who were assigned to both the project code-named "Dragon" and the one code-named "Lancelot." Figure 10 shows a rule that would retrieve all such people. Instead of listing all the technical job titles by name ("software engineer", "systems programmer", etc.), the rule includes an embedded description to determine whether a particular job title is on the technical, as opposed to the managerial or administrative, career ladder.

In addition to this general interface for database retrieval, we have also implemented a specialized feature in Object Lens for determining the recipients of messages. With this feature, descriptions (like that shown in the IF field of Figure 10) can be embedded in the To and cc fields of a message. Then, when the message is sent, these descriptions are automatically applied to all the Person objects in the local knowledge base and the resulting people are inserted in the To and cc fields. This feature allows senders to create distribution lists that are dynamically computed at message-sending time based on the current information about people in their data base (see Zloof, 1981 for a similar capability).

4.4 Hypertext

As noted above, it is a straightforward matter to use many of the features of a hypertext system in Object Lens (e.g., Halasz, Moran, & Trigg, 1987; Garrett, Smith, & Meyrowitz, 1986; Delisle & Schwartz, 1986). For instance, our system currently contains an object type called Text that displays only two fields: Name and Text. The Text field of a Text object can contain links to as many other objects as desired. For example, Figure 11 shows a sample Text object that contains links to people and bibliographic citations as well as to another Text object.

In addition to the usual benefits of hypertext systems, Object Lens derives additional benefits from its integration of hypertext with other database, messaging, and computational capabilities. For instance, in order to insert a link to another node in a hypertext system, a user must first find the node to which the link will be made. In Object Lens, the database retrieval capabilities described above can be used to automatically find objects (such as people or bibliographic citations) that satisfy certain criteria. Then links to these objects can be inserted into the text. One desirable feature found in some hypertext systems that is not yet included in Object Lens is the ability to show and follow the incoming links to an object. We would like to implement this capability as another action available on all objects.

Even though the relationship between Object Lens and previous hypertext systems is not the primary focus of this paper, it is interesting to observe that Object Lens appears to have some functionality in at least four of the seven areas that Halasz (1987) listed as being needed in the next generation of hypermedia systems (search and query, computational engines, collaborative work, and tailorability).

5. CONCLUSION

In this paper, we have described a system called Object Lens that integrates facilities for hypertext, object-oriented databases, electronic messaging, and rule-based agents. Using the basic primitives provided by this system, we believe it will be relatively easy to create a wide variety of cooperative work applications. We have shown several such applications here, and an important focus of our ongoing research will be to test the generality of the framework further by implementing more applications within it.

Object Lens is an example of a semiformal system, a system that represents knowledge in a way that both people and their computational agents can process intelligently. We believe that much of the power and flexibility of this system results from its choice of primitives (semistructured objects, customizable folders, and semiautonomous agents) and from the template-based interfaces that make these primitives both visible and changeable by inexperienced computer users.

REFERENCES

Conklin, J. (1987). Hypertext: An introduction and survey. *IEEE Computer*, vol. 20, no. 9, pp. 17-41.

Crowston, K. & Malone, T. W. (1988). Computational agents to support cooperative work. Working Paper No. 2008-88, Center for Information Systems Research, Massachusetts Institute of Technology, Cambridge, MA.

Delisle, N. & Schwartz, M. (1986). Contexts - a partitioning concept for hypertext. *ACM Transactions on Office Information Systems, 5* (2), 168-186.

Dittrich, D. & Dayal, U. (Eds.) (1986). *Proceedings of the International Workshop on Object-Oriented Database Systems,* Asilomar, CA.

Fikes, R. & Kehler, T. (1985). The role of frame-based representation in reasoning. *Communications of the ACM, 28,* 904.

Garrett, L. N., Smith, K. E., & Meyrowitz, N. (1986). Intermedia: Issues, strategies, and tactics in the design of a hypermedia document system. *Proceedings of the Conference on Computer-Supported Cooperative Work,* Austin, TX, December 3-5, 1986, 163-174.

Greif, I. (1988). Computer-supported cooperative work: Breakthroughs for user acceptance (Panel description), *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '88),* Washington, D. C., May 16-18, 1988.

Halasz, F. G. (1987). Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM, 31* (7), 836-855.

Halasz, F. G., Moran, T. P., & Trigg, R. H. (1987). NoteCards in a nutshell. *Proceedings of the 1987 ACM Conference of Human Factors in Computer Systems (CHI+GI '87),* Toronto, Ontario, April 5-9, 45-52.

Lai, K. Y. (1987). Essays on Object Lens: A tool for supporting information sharing. Unpublished M. S. thesis, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.

Lee, J. & Malone, T. W. (1988a). How can groups communicate when they use different languages? Translating between partially shared type hierarchies. *Proceedings of the ACM Conference on Office Information Systems*, Palo Alto, CA, March 23-25, 1988.

Lee, J. & Malone, T. W. (1988b). Partially Shared Views: A scheme for communicating among groups that use different type hierarchies. Sloan School of Management Working Paper, Massachusetts Institute of Technology, Cambridge, MA, September, 1988.

Malone, T. W., Grant, K. R., Lai, K. Y., Rao, R., & Rosenblitt, D. (1987a). Semistructured messages are surprisingly useful for computer-supported coordination. *ACM Transactions on Office Systems, 5,* 115-131.

Malone, T. W., Grant, K. R., Turbak, F. A., Brobst, S. A., & Cohen, M. D. (1987b). Intelligent information sharing systems. *Communications of the ACM, 30,* 390-402.

Malone, T. W., Grant, K. R., Lai, K. Y., Rao, R., & Rosenblitt, D. (in press). The Information Lens: An intelligent system for information sharing and coordination. In M. H. Olson (Ed.), *Technological support for work group collaboration*, Hillsdale, N. J.: Erlbaum.

Shriver, B. & Wegner, P. (1987). *Research directions in object-oriented programming*, Cambridge, MA: MIT Press.

Stefik, M. & Bobrow, D. G. (1986, Spring). Object-oriented programming: Themes and variations. *AI Magazine*, 40-62.

Sluizer, Suzanne & Cashman, P. M. (1984). XCP: An experimental tool for supporting office procedures. *IEEE 1984 Proceedings of the First International Conference on Office Automation,* Silver Spring, MD: IEEE Computer Society, 73-80.

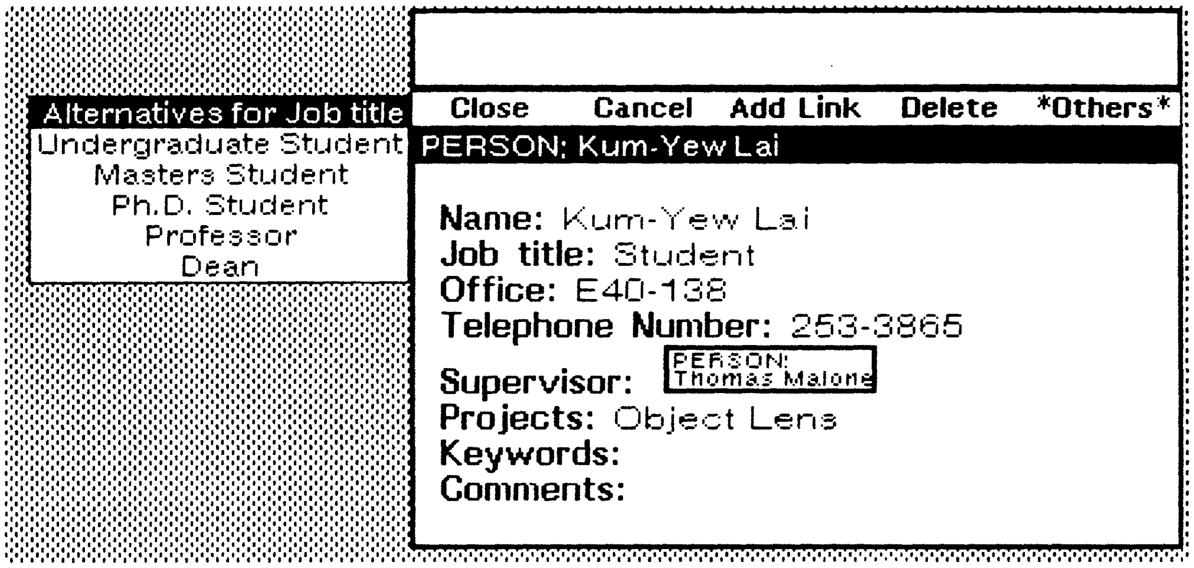
Tou, F. N., Williams, M. D., Fikes, R. E., Henderson, D. A., & Malone, T. W. (1982). RABBIT: An intelligent database assistnat. *Proceedings of the National Conference of the American Association of Artificial Intelligence,* Pittsburgh, Pennsylvania, August 18-20.

Turbak, F. A. (1986). *Grasp: A visible and manipulable model for procedural programs.* Unpublished M. S. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Winograd, T. (1988). A language/action perspective on the design of cooperative work. *Human Computer Interaction, 3* (1), 3-30.

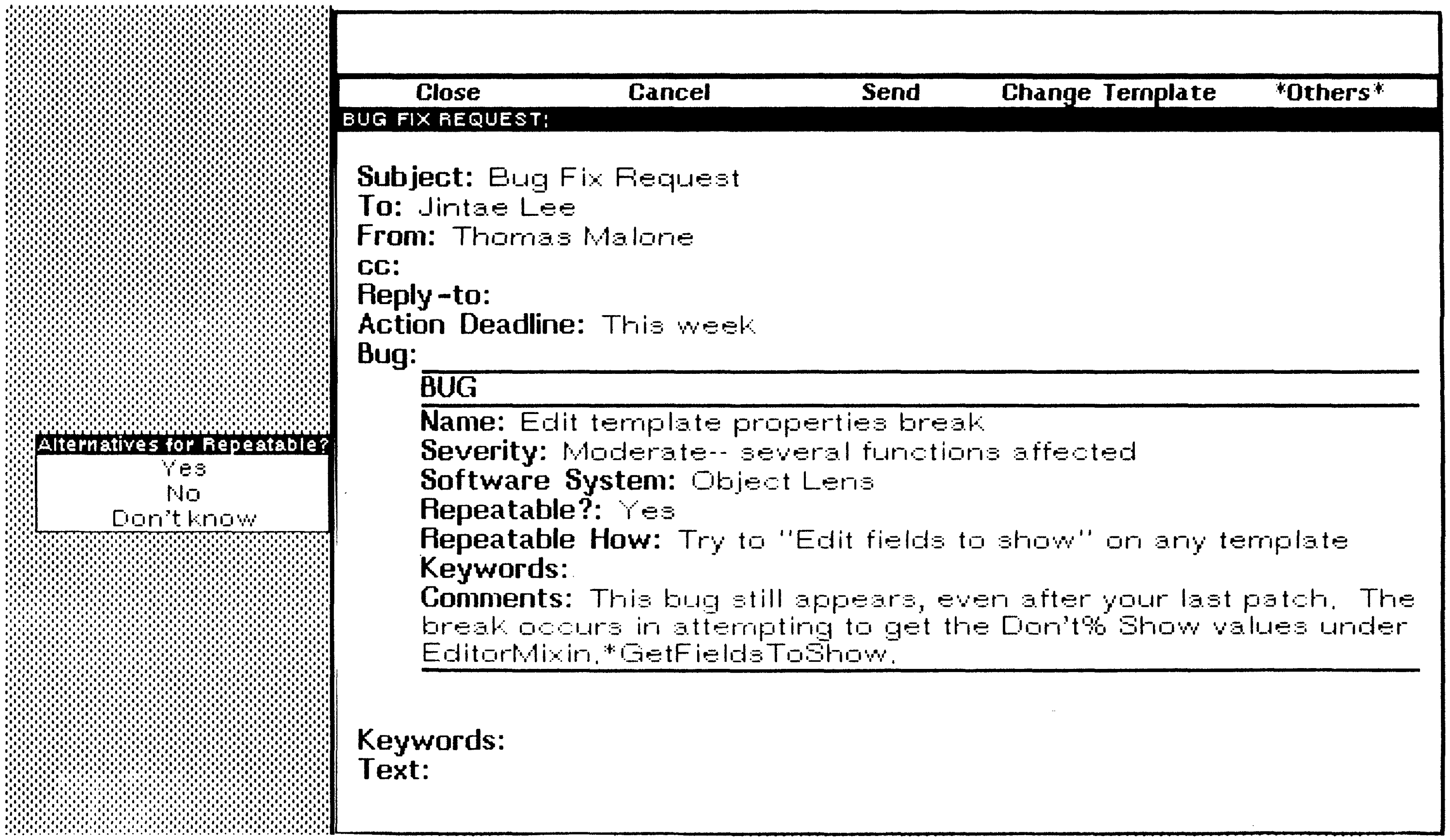
Winograd, T. & Flores, F. (1986). *Understanding computers and cognition: A new foundation for design.* Norwood, NJ: Ablex.

Zloof, M. M. (1981). QBE/OBE: A language for office and business automation. *IEEE Computer 14* (May), 5.



Name: Kum-Yew Lai
Job title: Student
Office: E40-138
Telephone Number: 253-3865
Supervisor: PERSON: Thomas Malone
Projects: Object Lens
Keywords:
Comments:

Figure 1. Objects can be edited with a simple template editor. Fields can include text, graphics, or links to other objects.



Subject: Bug Fix Request
To: Jintae Lee
From: Thomas Malone
cc:
Reply-to:
Action Deadline: This week
Bug:

BUG
Name: Edit template properties break
Severity: Moderate-- several functions affected
Software System: Object Lens
Repeatable?: Yes
Repeatable How: Try to "Edit fields to show" on any template
Keywords:
Comments: This bug still appears, even after your last patch. The break occurs in attempting to get the Don't% Show values under EditorMixin.*GetFieldsToShow.

Keywords:
Text:

Figure 2. Embedded templates allow related objects to be viewed and edited simultaneously.

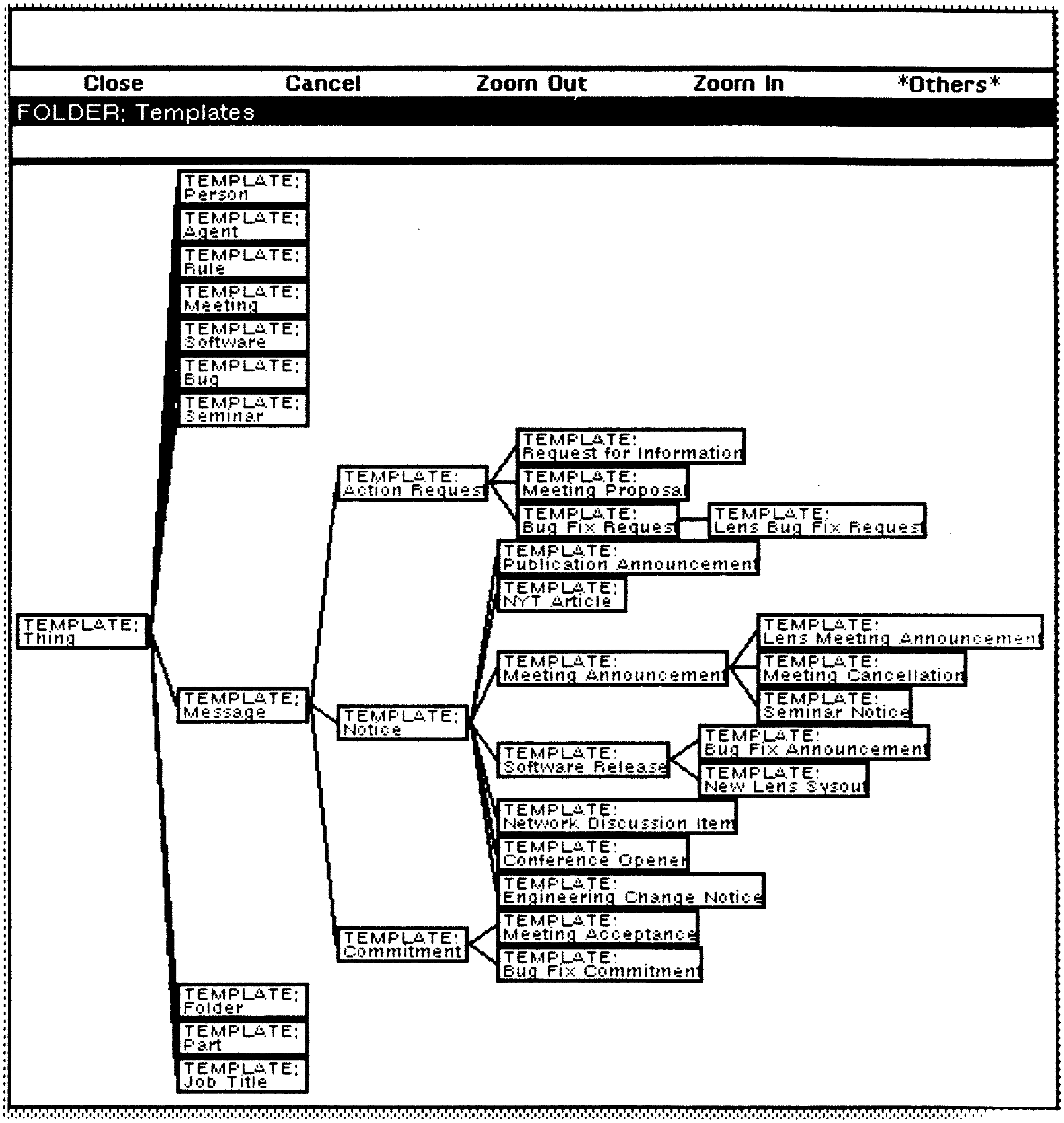
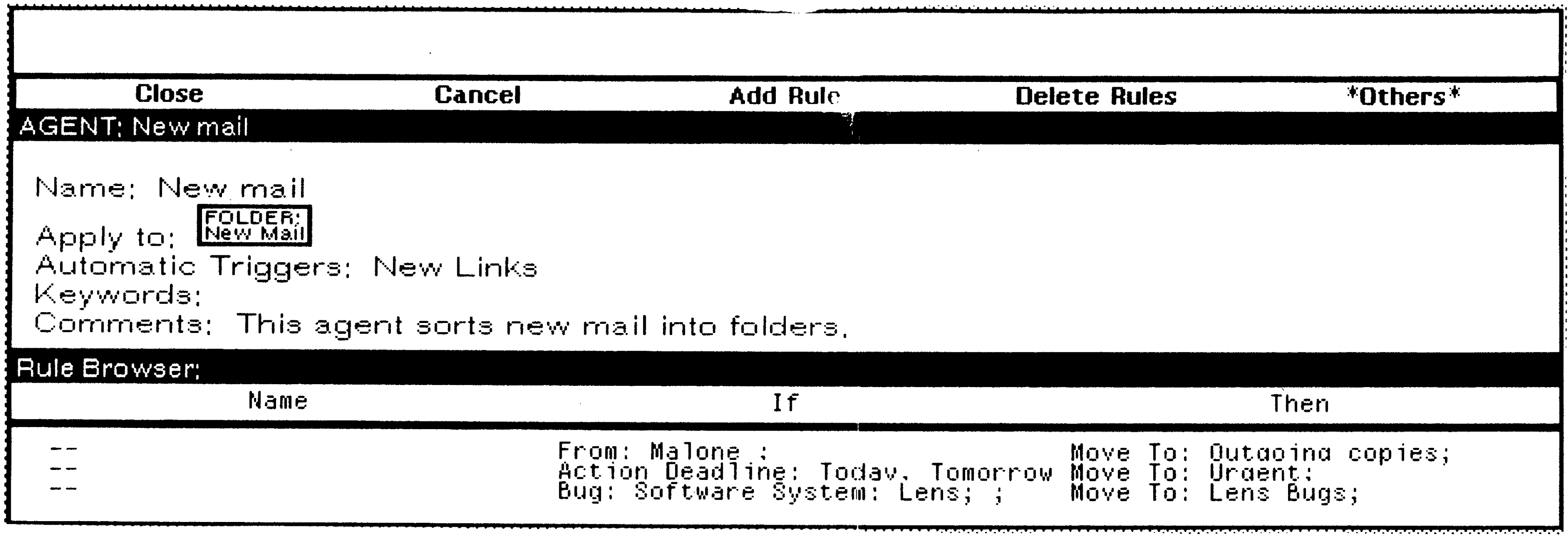

Close
Cancel
Zoom Out
Zoom In
Others
FOLDER: Templates
TEMPLATE: Thing
TEMPLATE: Person
TEMPLATE: Agent
TEMPLATE: Rule
TEMPLATE: Meeting
TEMPLATE: Software
TEMPLATE: Bug
TEMPLATE: Seminar
TEMPLATE: Message
TEMPLATE: Action Reques
TEMPLATE: Request for Information
TEMPLATE: Meeting Proposal
TEMPLATE: Bug Fix Reques
TEMPLATE: Lens Bug Fix Reques
TEMPLATE: Notice
TEMPLATE: Publication Announcement
TEMPLATE: NYT Article
TEMPLATE: Meeting Announcement
TEMPLATE: Lens Meeting Announcement
TEMPLATE: Meeting Cancellation
TEMPLATE: Seminar Notice
TEMPLATE: Software Release
TEMPLATE: Bug Fix Announcement
TEMPLATE: New Lens Sysout
TEMPLATE: Network Discussion Item
TEMPLATE: Conference Opener
TEMPLATE: Engineering Change Notice
TEMPLATE: Commitment
TEMPLATE: Meeting Acceptance
TEMPLATE: Bug Fix Commitment
TEMPLATE: Folder
TEMPLATE: Part
TEMPLATE: Job Title

Figure 3. Object types are defined by a set of Templates.



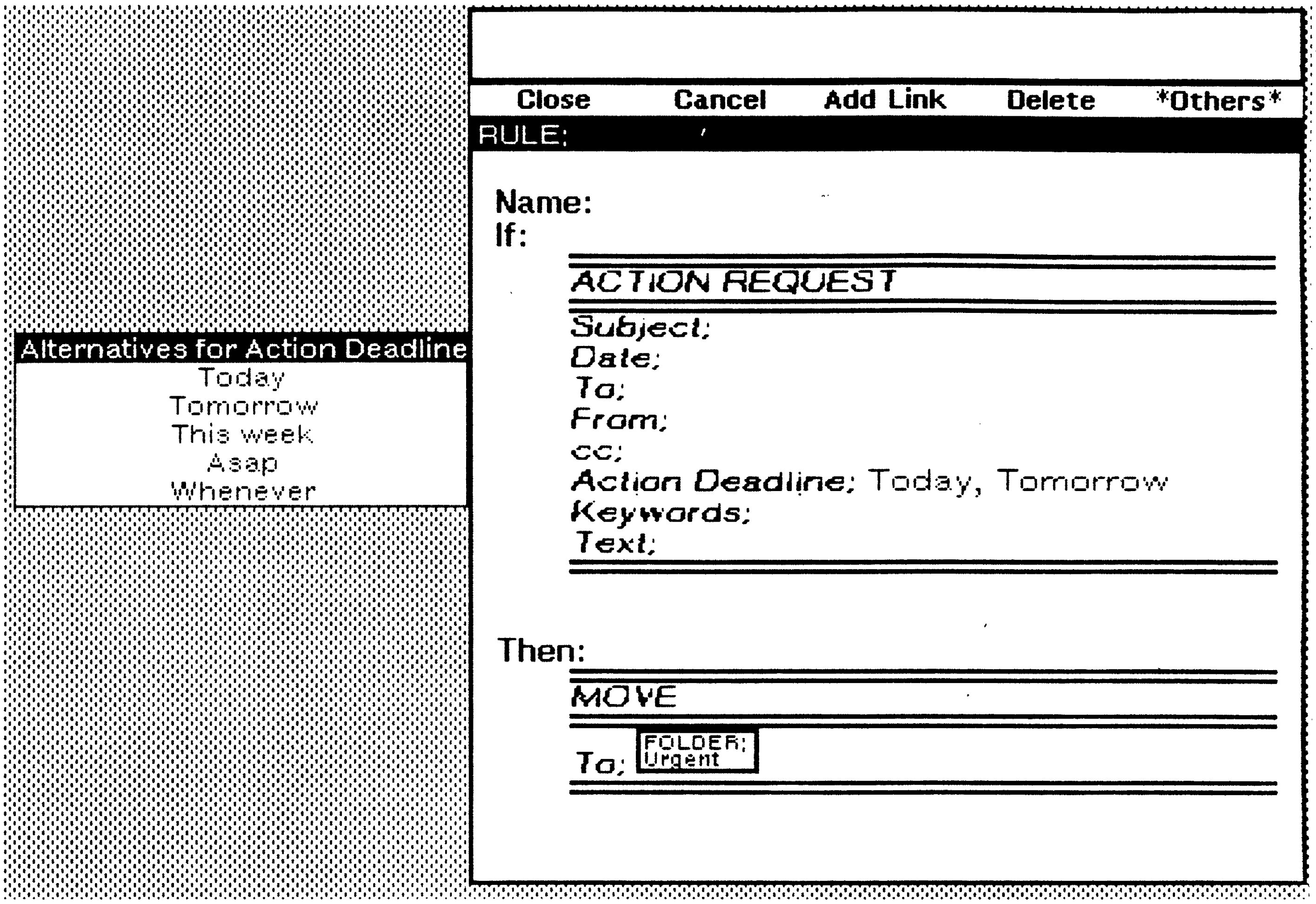
AGENT: New mail

Name: New mail
Apply to: FOLDER: New Mail
Automatic Triggers: New Links
Keywords:
Comments: This agent sorts new mail into folders.

Rule Browser:

Name	If	Then
--	From: Malone ;	Move To: Outgoing copies;
--	Action Deadline: Today, Tomorrow	Move To: Urgent;
--	Bug: Software System: Lens; ;	Move To: Lens Bugs;

Figure 4. Agents include a collection of rules and specifications for when and where to apply them.



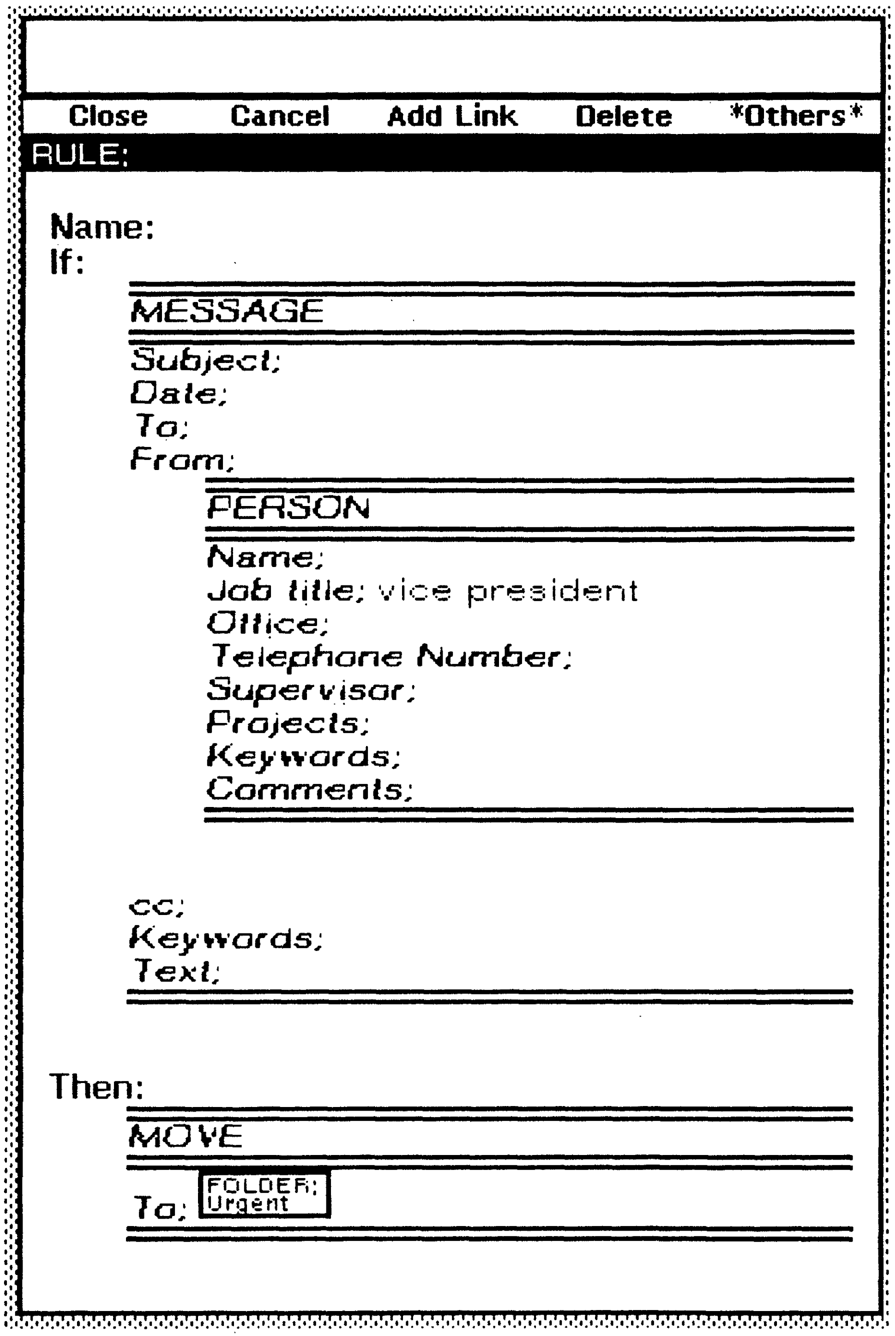
Name:
If:

ACTION REQUEST
Subject:
Date:
To:
From:
cc:
Action Deadline: Today, Tomorrow
Keywords:
Text:

Then:

MOVE
To: FOLDER: Urgent

Figure 5. Rules describe the objects that will satisfy them and specify what action to perform on those objects.



Name:
If:

MESSAGE

Subject;
Date;
To;
From;

PERSON

Name;
Job title; vice president
Office;
Telephone Number;
Supervisor;
Projects;
Keywords;
Comments;

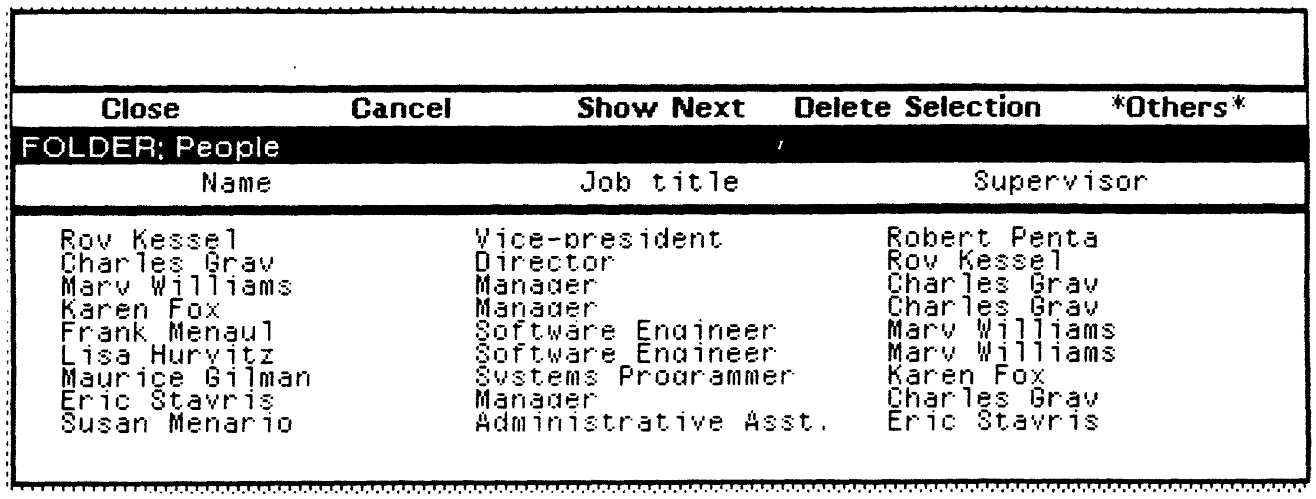
cc;
Keywords;
Text;

Then:

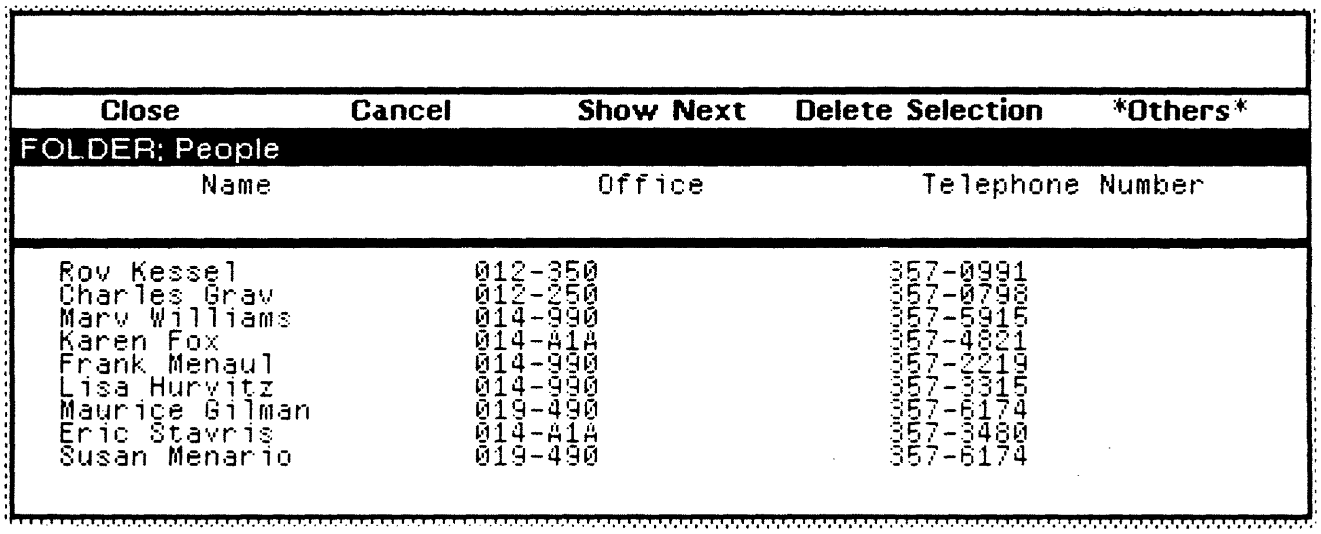
MOVE

To; FOLDER; Urgent

Figure 6. Rules can use embedded descriptions to create complex queries.



Name	Job title	Supervisor
Rov Kessel	Vice-oresident	Robert Penta
Charles Grav	Director	Rov Kessel
Marv Williams	Manaoer	Charles Grav
Karen Fox	Manaoer	Charles Grav
Frank Menaul	Software Enaineer	Marv Williams
Lisa Hurvitz	Software Enaineer	Marv Williams
Maurice Gilman	Svstems Prooranmer	Karen Fox
Eric Stavris	Manaoer	Charles Grav
Susan Menario	Administrative Asst.	Eric Stavris



Name	Office	Telephone Number
Rov Kessel	012-350	357-0991
Charles Grav	012-250	357-0798
Marv Williams	014-990	357-5915
Karen Fox	014-A1A	357-4821
Frank Menaul	014-990	357-2219
Lisa Hurvitz	014-990	357-3315
Maurice Gilman	019-490	357-6174
Eric Stavris	014-A1A	357-3480
Susan Menario	019-490	357-6174

Figures 7 a,b,c. Users can select which fields to display in tables that summarize a collection of objects.

Figures 7a & 7b

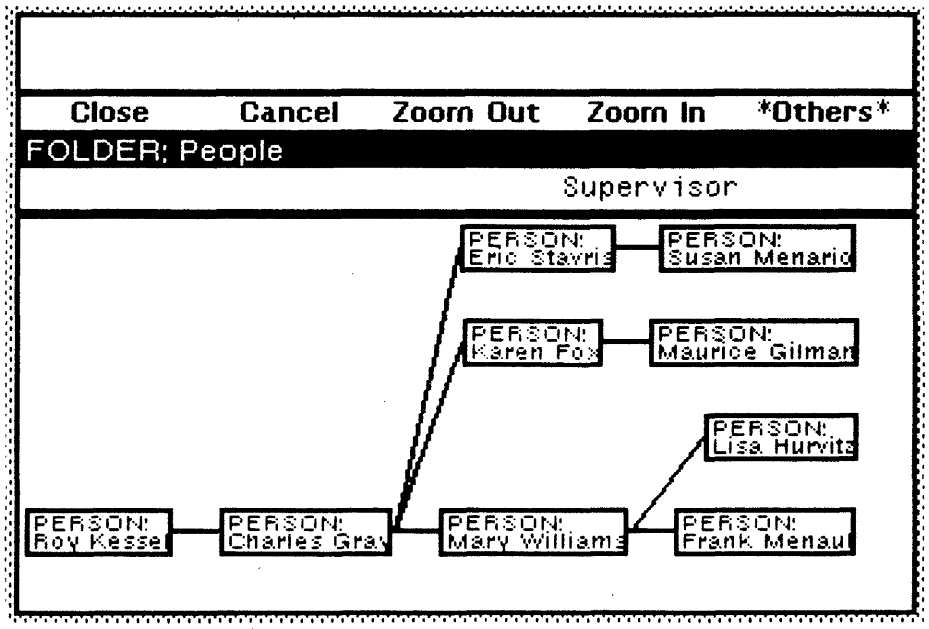
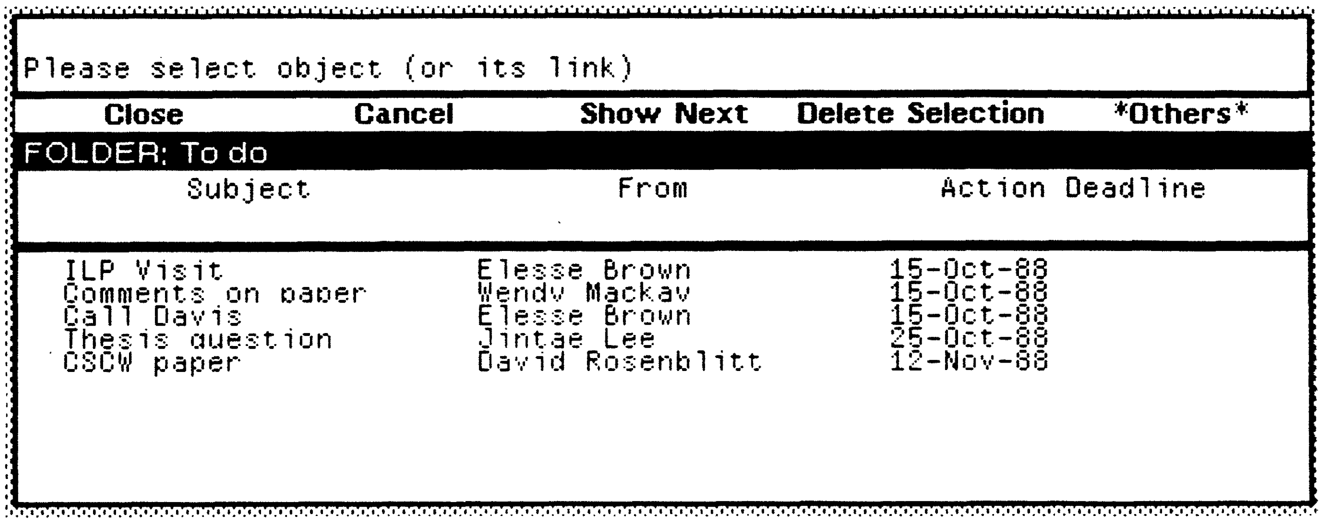

Close
Cancel
Zoom Out
Zoom In
Others
FOLDER: People
Supervisor
PERSON: Eric Stavris
PERSON: Susan Menario
PERSON: Karen Fox
PERSON: Maurice Gilman
PERSON: Lisa Hurvitz
PERSON: Roy Kessel
PERSON: Charles Gray
PERSON: Mary Williams
PERSON: Frank Menaul

Figure 7c



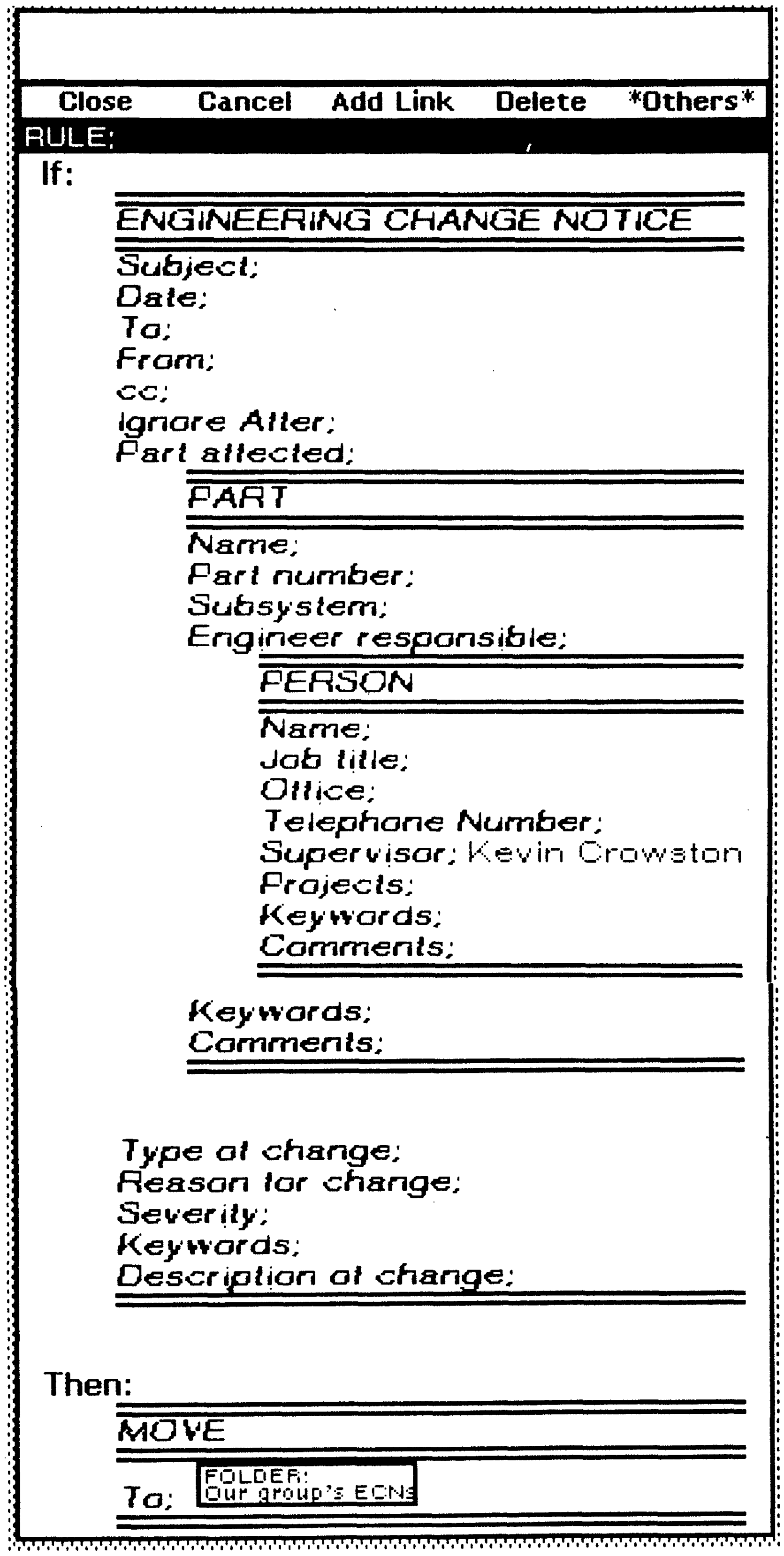
Please select object (or its link)

Close | Cancel | Show Next | Delete Selection | *Others*

FOLDER: To do

Subject	From	Action Deadline
ILP Visit	Elesse Brown	15-Oct-88
Comments on paper	Wendy Mackay	15-Oct-88
Call Davis	Elesse Brown	15-Oct-88
Thesis question	Jintae Lee	25-Oct-88
CSCW paper	David Rosenblitt	12-Nov-88

Figure 8. Tables can be used to summarize selected fields from Action Request messages.



RULE:

If:

ENGINEERING CHANGE NOTICE

Subject;
Date;
To;
From;
cc;
Ignore After;
Part affected;

PART

Name;
Part number;
Subsystem;
Engineer responsible;

PERSON

Name;
Job title;
Office;
Telephone Number;
Supervisor; Kevin Crowston
Projects;
Keywords;
Comments;

Keywords;
Comments;

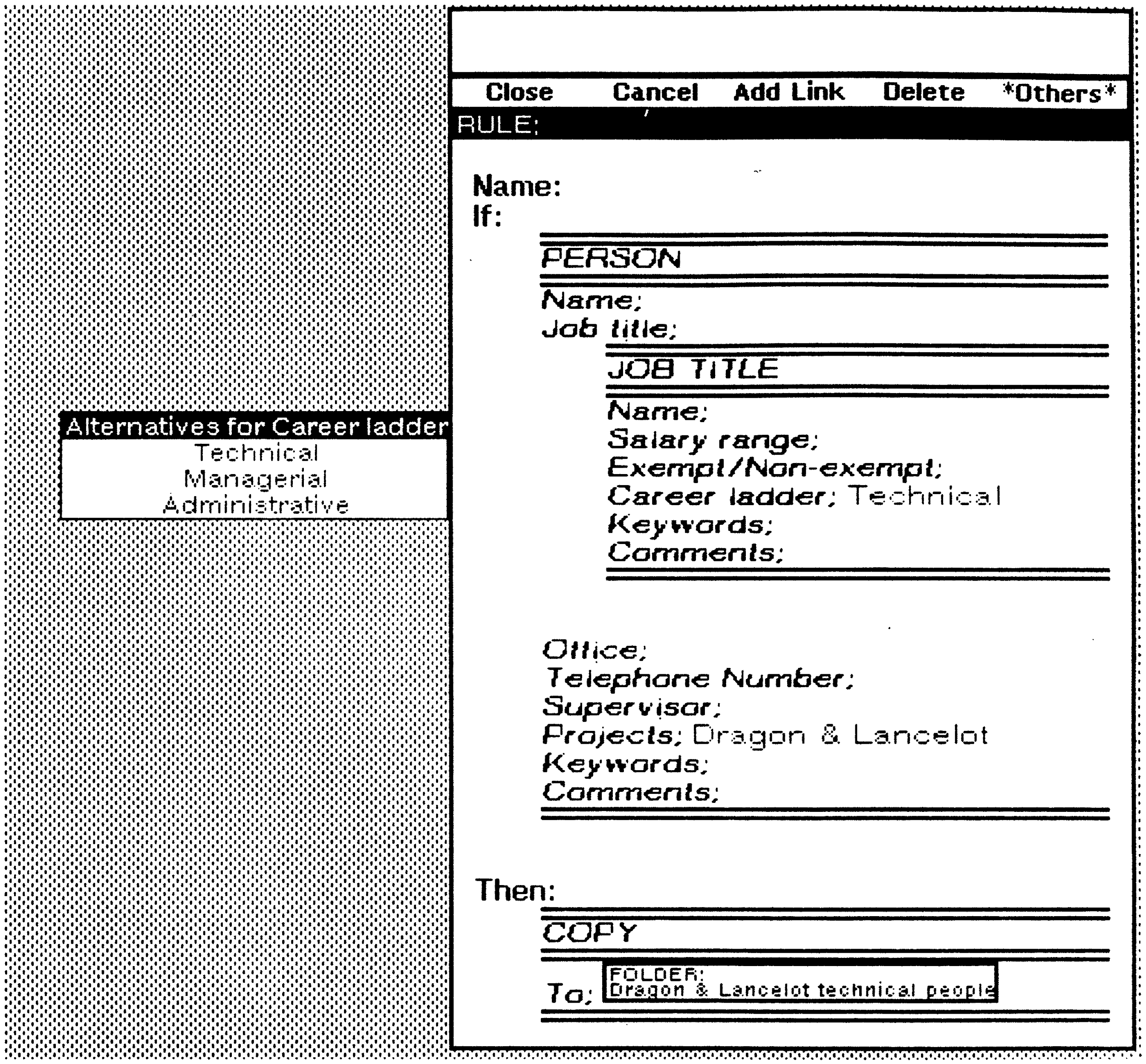
Type of change;
Reason for change;
Severity;
Keywords;
Description of change;

Then:

MOVE

To; FOLDER: Our group's ECNs

Figure 9. Rules can include multiple levels of embedded descriptions that refer to linked objects throughout the knowledge base.



RULE:

Name:
If:

PERSON

Name;
Job title;

JOB TITLE

Name;
Salary range;
Exempt/Non-exempt;
Career ladder; Technical
Keywords;
Comments;

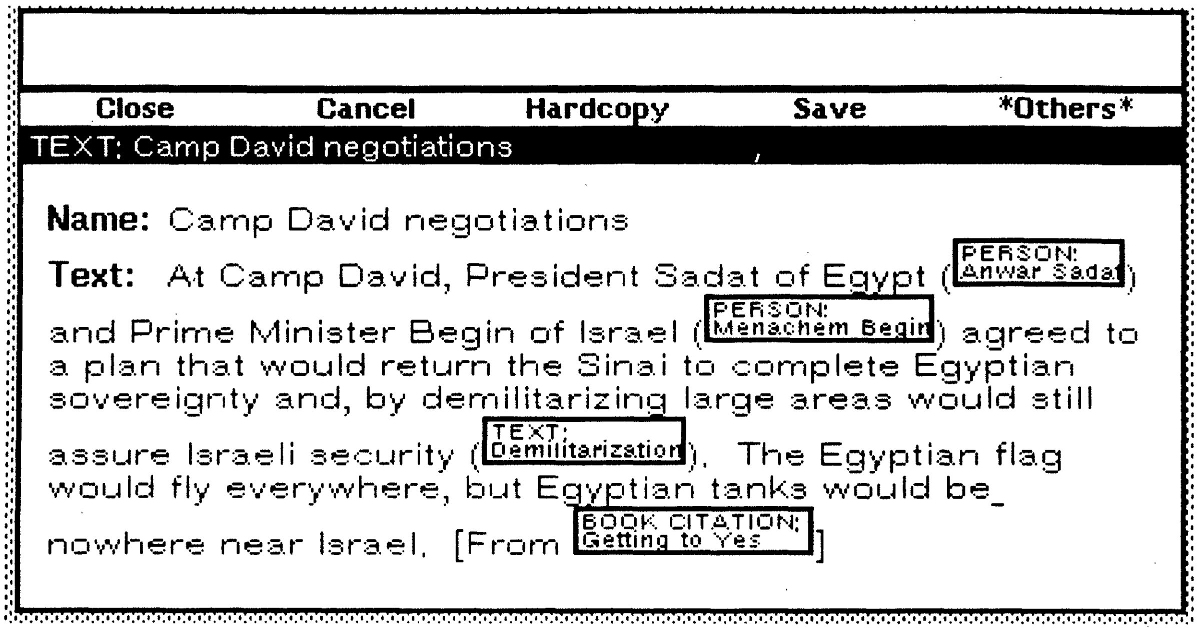
Office;
Telephone Number;
Supervisor;
Projects; Dragon & Lancelot
Keywords;
Comments;

Then:

COPY

To; FOLDER: Dragon & Lancelot technical people

Figure 10. Agents can retrieve all the objects from a database that satisfy certain criteria.



Name: Camp David negotiations

Text: At Camp David, President Sadat of Egypt (PERSON: Anwar Sadat) and Prime Minister Begin of Israel (PERSON: Menachem Begin) agreed to a plan that would return the Sinai to complete Egyptian sovereignty and, by demilitarizing large areas would still assure Israeli security (TEXT: Demilitarization). The Egyptian flag would fly everywhere, but Egyptian tanks would be nowhere near Israel. [From BOOK CITATION: Getting to Yes]

Figure 11. Hypertext documents can include links, not only to other text passages, but also to other object types such as people and bibliographic citations.

Management in the 1990s
Working Paper Series

90s No.	Title	Author(s)	Date
85-001	Management in the 1990s Mission Statement	---------	7/85
85-002	Sponsors' Vision Document	----------	5/85 (Rev 1/86)
85-003	Management in the 1990s Program Research Themes and Request for Proposals, Fall 1985	---------	8/85
85-004	A Survey of Current Trends in the Use of Executive Support Systems	DeLong Rockart	11/84
85-005	Expert Systems and Expert Support Systems: The Next Challenge for Management	Luconi Malone Scott Morton	9/85
85-006	Future Directions in DSS Technology	Treacy	1/85
85-007	Information Technology and Corporate Strategy: A Research Perspective	Bakos Treacy	3/85 (Rev 3/86)
85-008	Supporting Senior Executives' Models for Planning and Control	Treacy	6/85
85-009	Designing Organizational Interfaces	Malone	9/85
85-010	Out of print. Superceded by 86-025		
85-011	Organizational Structure andTechnology: Elements of a Formal Theory	Malone	8/85
85-012	Computer Support for Organizations: Toward an Organizational Science	Malone	9/85
85-013	Influence of Task Type on the Relation Between Communication and Performance	Hauptman	5/85
85-014	Implications of Changes in Information Technology for Corporate Strategy	Scott Morton Rockart	1/83 (Rev 3/84)
86-015	Identifying the Attributes of a Successful Executive Support System Implementation	De Long Rockart	2/86
86-016	Out of print. Superceded by 86-025		

90s No.	Title	Author(s)	Date
86-017	Information Technology, Integration, and Organizational Change	Benjamin Scott Morton	4/86
86-018	Electronic Markets and Electronic Hierarchies: Effects of Information Technology on Market Structure and Corporate Strategy	Benjamin Malone Yates	4/86
86-019	Toward a Cumulative Tradition of Research on Information Technology as a Strategic Business Factor	Treacy	4/86
86-020	Executive Support Systems and the Nature of Executive Work	Rockart DeLong	4/86
86-021	Electronic Organization and Expert Networks: Beyond Electronic Mail and Computer Conferencing	Stevens	5/86
86-022	Out of print. Superceded by 87-032		
86-023	Economic Issues in Standardization	Farrell Saloner	10/85
86-024	Floor Lay-out Design for Effective Software Production: Applying the Implications from the Optimal Communication Pattern of a Software Project Team	Hauptman	6/86
86-025	Intelligent Information Sharing Systems	Malone Grant Turbak Brobst	11/86
86-026	Semi-Structured Messages are Surprisingly Useful for Computer-Supported Coordination	Malone Grant Lai Rao Rosenblitt	11/86
86-027	Cognitive Science and Organizational Design: A Case Study of Computer Conferencing	Crowston Malone Lin	10/86
86-028	Strategy Formulation Methodologies	Scott Morton	11/86
86-029	Some Thoughts on the Information Technology Revolution in Standard Oil	Horton	11/86

90s No.	Title	Author(s)	Date
86-030	The Value-Added of Strategic IS Planning: Understanding Consistency, Validity, and IS Markets	Henderson Sifonis	11/86
86-031	A Common Agenda for an Uncommon Future: Address to the Sloan School of Management	Whitmore	11/86
87-032	Managing the Data Resource: A Contingency Perspective	Goodhue Quillard Rockart	2/87
87-033	Information Technology in Marketing	Little	2/87
87-034	Finding Synergy Between Decision Support Systems and Expert Systems Research	Henderson	3/87
87-035	Data Envelopment Analysis for Managerial Control and Diagnosis	Epstein Henderson	5/87
87-036	Managing the IS Design Environment: A Research Framework	Henderson	5/87
87-037	Freeing Work from the Constraints of Location and Time: An Analysis Based on Data from the United Kingdom	Bailyn	6/87
87-038	The Influence of Communication Technologies on Organizational Structure: A Conceptual Model for Future Research	Hauptman Allen	5/87
87-039	The Line Takes the Leadership	Rockart	8/87
87-040	Information Technology and Work Organization	Crowston Malone	10/87
88-041	Out of print. Superceded by 88-055.		
88-042	The Realities of Electronic Data Interchange: How Much Competitive Advantage?	Benjamin DeLong Scott Morton	1/88
88-043	Determinants of Employees' Affective Response to the Use of Information Technology in Monitoring Performance	Chalykoff	2/88
88-044	How Expectations About Microcomputers Influence Their Organizational Consequences	Carroll Perin	4/88

90s No.	Title	Author(s)	Date
88-045	Toward the Perfect Workplace? The Experience of Home-Based Systems Developers.	Bailyn	3/88
88-046	The Impact of Technological Change on Employment Structure in Telecommunications: Whatever Happened to the Wichita Lineman?	Lynch Osterman	4/88 (Rev. 9/88)
88-047	The Competitor Intelligence Function in the Very Large-Scale Organization: Assessments and Uses	Westney Ghoshal	5/88
88-048	Adding Value in an Information Function	Westney Ghoshal	5/88
88-049	Compatibility Standards and the Market for Telecommunications Services	Besen Saloner	5/88
88-050	Assessing IT Performance: What the Experts Say	Wilson	6/88
88-051	The Moral Fabric of the Office: Organizational Habits vs. High-Tech Options for Work Schedule Flexibilities	Perin	6/88
88-052	Employment Security at DEC: Sustaining Values Amid Environmental Change	Kochan MacDuffie Osterman	6/88
88-053	An Economic Study of the Information Technology Revolution	Jonscher	6/88
88-054	An Assessment of the Productivity Impact of Information Technologies	Loveman	8/88
88-055	Involvement as a Predictor of Performance in I/S Planning and Design	Henderson	8/88
88-056	Planning and Managing Change	Schein	8/88 (Rev 10/88)
88-057	Information Technology: From Impact On to Support For Organizational Design	Invernizzi	9/88
88-058	Information Technology and the New Organization: Towards More Effective Management of Interdependence	Rockart Short	9/88

90s No.	Title	Author(s)	Date
88 059	Dimensions of I/S Planning and Design Technology	Henderson Cooprider	9/88
88 060	Issues of Gender in Technical Work	Bailyn	10/88
88 061	More than Just a Communication System: Diversity in the Use of Electronic Mail	Mackay	10/88
88 062	Managing New Technology and Labor Relations: An Opportunity for Mutual Influence	McKersie Walton	10/88
88 063	Information Technology, Human Resource Management, and Organizational Learning: A Case Study in Telecommunications	Villiger	10/88
88 064	Innovative Cultures and Organizations	Schein	11/88
89 065	Computer-Aided Monitoring: Its Influence on Employee Job Satisfaction and Turnover	Chalykoff Kochan	1/89
89 066	Partially Shared Views: A Scheme for Communicating among Groups that Use Different Type Hierarchies	Lee Malone	3/88
89 067	What is Coordination Theory?	Malone	2/88
89 068	Joint Ventures in the Information Technology Sector: An Assessment of Strategies and Effectiveness	Koh Venkatraman	12/88
89 069	Implementation of End User Computing in the Internal Revenue Service	Pentland	3/89
89 070	A Process Model of Strategic Alliance Formation in Firms in the Information Technology Industry	Wilson	3/89
89 071	Object Lens: A "Spreadsheet" for Cooperative Work	Lai Malone Yu	9/88
89 072	Electronic Integration and Strategic Advantage: A Quasi-Experimental Study in the Insurance Industry	Venkatraman Zaheer	4/89
89 073	Electronic Mail and Organizational Cooperation: A Case Study	Invernizzi Luberto	6/89
89 074	Cotechnology for the Global 90s	Stevens	7/89