ESSAYS ON OBJECT LENS, A TOOL FOR SUPPORTING INFORMATION-SHARING

by

KUM-YEW LAI

S.B., Massachusetts Institute of Technology
(1987)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN MANAGEMENT

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1987

Signature of Author: _____
Alfred P. Sloan School of Management
August 14, 1987

Certified by: _____
Professor Thomas W. Malone
Thesis Supervisor

Accepted by: _____
Jeffrey A. Barks
Associate Dean, Master's and Bachelor's Program

ESSAYS ON OBJECT LENS, A TOOL FOR SUPPORTING INFORMATION-SHARING

by

Kum-Yew Lai

Submitted to the Alfred P. Sloan School of Management
on August 22, 1987, in partial fulfillment of the requirements
for the Degree of Master of Science in Management

ABSTRACT

There exists the technology today to build large-scale knowledge bases, hypertext systems, as well as intelligent information sharing systems. As these three kinds of technologies become more widely used, the need to integrate them into generic office tools becomes more urgent for two reasons. First, each technology by itself has its limitations that will become increasingly burdensome. Second, their integration produces a synergy that can overcome their individual limitations.

The goal of this paper is to articulate the synergy that arises from systems that integrate these separate technologies and to describe an implementation of such a system called Object Lens.

Thesis Supervisor: Dr. Thomas W. Malone
Title: Douglas Drane Career Development
Associate Professor of Information Technology and Management

TABLE OF CONTENTS

ACKNOWLEDGMENTS

The idea of working on an object-oriented version of the Lens system has been mooted more than a year ago. Along the way, until the system is implemented, many people have contributed significantly. First and foremost is my supervisor, Professor Thomas Malone. Tom has been my mentor since my undergraduate days. He has also been a immense source of ideas, support, and encouragement. I cannot thank him enough. Randy Davis kindly agreed to be my thesis reader; I only regret I do not have the good sense to learn more from him.

Others in the Lens group have contributed to this project. Ken Grant is "the Legend," who is one of the first to spawn ideas about Object Lens. Kevin Crowston and Jintae Lee are friends, colleagues, and unending sources of ideas. Cheryl Clark, Wendy Mackay, David Rosenblitt, and Franklyn Turbak give many constructive comments during the initial stages.

From Xerox's Palo Alto Research Center, Stan Lanning, Ramana Rao, and Randy Trigg helped with ideas on the implementation.

Finally, the Development Bank of Singapore sponsored my studies here, for which I am grateful.

GENERALIZED TOOLS FOR COOPERATIVE WORK

ABSTRACT

There exists the technology today to build large-scale knowledge bases, hypertext systems, as well as intelligent information sharing systems. As these three kinds of technologies become more widely used, the need to integrate them into generic office tools becomes more urgent for two reasons. First, each technology by itself has its limitations that will become increasingly burdensome. Second, their integration produces a synergy that can overcome their individual limitations..

The aim of this paper is to articulate the synergy that arises from systems that integrate these separate technologies and to describe an implementation of such a system called Object Lens.

INTRODUCTION

Many of the existing technologies used to support cooperative office work are developed separately from each other. As these technologies become more mature and economically feasible, their widespread use will reveal their individual limitations. Three of these stand out among others as most useful and yet limited in their isolated form. These are hypertext systems, knowledge bases, and office coordination tools.

Hypertext systems have been developed primarily for either of the following goals:

- to browse and present complex information spaces (*e.g.* [Gar86]).

- to support idea structuring, especially in the initial stages of problem solving (*e.g.* [Tri86]).

However, these goals limit hypertext systems to a passive mode with no possibility for the automatic processing of the stored information. The need for the

machine processing of hypertext will become more urgent for at least two reasons. First, it will be economically practical to build and maintain large-scale hypertext systems, Second, hypertext systems are becoming more widely used. For example, Apple Corporation recently released its commercial hypertext system called HyperCard.

Much of the advances in the machine processing of information have been in the area of knowledge-based systems. The structured representation of knowledge is a key to such systems. However, incorporating the rigid nature of structured representations into hypertext would greatly reduce the usefulness hypertext. Semi-structured representations [Mal87b] are more amenable. They can make use of many techniques in artificial intelligence, such as inheritance and production rules, without limiting the usefulness of hypertext. Indeed, as knowledge-based systems themselves become more widely used, semi-structured knowledge representations can also become more important for less structured domains like office work.

On another plane, advances in communications technology will make intelligent information sharing systems increasingly important [Mal87a]. A major goal of such systems is to support increased information metabolism in a community without inudating each person with less relevant information – helping each person to get the information she wants, no more (*e.g.* by automatically filtering out less useful information) or less (*e.g.* by help the user get information she would not otherwise have seen). Lens [Mal87a] is such a system; and it is also based on semi-structured representations of messages.

Integrating an information sharing system with an information storage system results in a system much more powerful than if each system were to be used separately. For example, for the receiver, the information communicated in systems like Lens can be filtered not just on the basis of what is in the messages, but also what is already known by the receiver. Conversely, the messages can automatically be used to update the receiver's information. For the sender, the information stored can help her compose her messages (*e.g.* she can get help on deciding who to send her messages). We have earlier argued that hypertext systems are powerful and increasing widely used tools for storing and browsing information. Therefore, a natural step to create a more general tool for cooperative work is to combine intelligent information sharing systems with hypertext.

OBJECT LENS: A GENERALIZED TOOL FOR SUPPORTING COOPERATIVE WORK

Object Lens is a generalized tool that supports many of the activities in cooperative work, such as information sharing, as well as activities in office work, such as retrieving and browsing complex information. It uses many of the ideas in Lens, and generalized them with a hypertext system based on semi-structured representations.

Object Lens is based on the following key ideas:

• a set of semi-structured templates can form the basis of a generalized office system. With these templates, users can create objects in the system that represent many physically or conceptually familiar things such as messages, persons, meetings, manufactured parts, and software bugs. For example, the SOFTWARE BUG template may have fields like Software System Affected:, Bug Fixer:, and Deadline for Fix:. Besides being easy to identify with, these templates help people to construct objects in the system because each field prompts the user to fill in the appropriate values. These values may be other objects or just free text. A key requisite in the idea of semi-structuredness is that users can fill in as much or as little information as possible, and the information filled can be as structured (*e.g.* they are also objects, with fields and values) or as unstructured (*e.g.* free text like "I don't really know") as possible.

Semi-structured templates not only help people to construct objects, but they also help the system to automatically process them. Of course, the system can only process the objects to the extent that they have expected values (*e.g.* the Deadline for Fix: of a SOFTWARE BUG object is filled with a date); however, it does not penalize the user or breaks down if the values cannot be processed (*e.g.* the Deadline for Fix: is filled with "anytime, buddy"). This graceful degradation is important in the use of semi-structured representations.

• links to objects form the basis of the hypertext system. Object can contain links to other objects. This mechanism forms a general framework so that users can associated an object with other objects. This association network is straightforward to build, and yet can be processed by the same mechanisms used for processing "flat" objects (objects without links to other objects). Also, the links

allow the user to be concerned with objects, and not where they could appear. For instance, the user could have two links to an object, each in a different folder. Changing the underlying object will allow the user to see the changed object regardles of which link she uses to access the object.

• descriptions allow users to specify arbitrary sets of objects. Descriptions have the same templates as objects. For example, a MESSAGE DESCRIPTION has the same template as a MESSAGE object. By filling values in a description, the user restricts the set of objects that are considered to satisfy the description. In the MESSAGE DESCRIPTION example, if the user fills in the To: field with "Kum-Yew Lai", the that description represents all those MESSAGE objects with "Kum-Yew Lai" as one of the values in their To: fields. A user can also put descriptions as values in descriptions, hence creating arbitrarily complex representations of sets of objects.

• programmable agents form the basic of automatic processing of information. Agents in Object Lens are also a kind of objects. Each of them contains a set of production rules (each of which is in turn an object), which specify what the agent should do when it is processing an object. Two examples of the actions that agents can do are updating the object being processed and call another agent to do *its* processing. Agents remain dormant until they are triggered by events like new mail coming into a folder, or a specified time of the day, or by the user or other agents. These agents can also reside on remote workstations and act as information brokers for the user.

• a consistent and customizable user interface is used to for browsing and editing objects and templates. This allows the user to view her information space in very flexible ways. For example, folders contain objects; these objects can be displayed in a network or a table. In a particular case, the folder of templates can be displayed in a frame inheritance network.

• graceful degradation and incrementality help increase the acceptability of the system. Graceful degradation prevents the system from breaking down even if the user does not do everything necessary for it to function at its best. For example, it is best that the user can fill in expected values for each field in an object (*e.g.* a date for the Deadline: field, as opposed to "I don't really know" or leaving it blank). Even if the user does not do so, the system can still work, although it does so t a suboptimal level (*e.g.* process other fields of the object and ignore the Deadline:

field). Conversely, the user need not do all that is required to make full use of the system. One example is the same Deadline: field above – the user (or her addressees) just do not get the optimal processing of the object. As an another example, users as a group can make use of shared templates to get the full benefit of Object Lens. Even if this is not possible, individual users can get some benefit by making use of their local objects world (*e.g.* to help process their messages. to use messages to update the objects world, to use the hypertext objects world for keeping their documents). And finally, users who do not want to use any objects do not suffer if other users do so. For example, messages in Object Lens can be read by any other kinds of mail system.

The above are the primary features of the Object Lens system. In the following sections, we discuss each of them in turn. The appendix contains a section which describes all the features of Object Lens in detail.

SEMI-STRUCTURED TEMPLATES PROVIDE THE BASIS FOR THE OBJECTS WORLD

The templates in Object Lens are in a system folder called User Templates; Figure 1. The figure shows the links to the templates. Presenting these links in a network makes it easier for the user to find a desired template. In the network, a subtemplate inherits the fields from its parent template. For example, the REQUEST template has a field called Deadline: which is not in the MESSAGE template. Each field as four aspects, and these are also inherited along the template network. The four aspects of each field are: the default value, an explanation, a set of alternatives that can be used as values of the field, and a set of templates from which the user can create descriptions. These are described below.

At this point, we make a digression to desribe the notational convention used in this paper, and to describe what FOLDER objects are. We use helvetica font to present all parts of the Object Lens world – like objects, fields, and templates. Template names are presented in upper case, and other names (*e.g.* field names and object names) in lower case.

FOLDER objects, like other objects, have a template structure. However, they have only one field called Contents:. This contains any number of links to objects or templates. For instance, the User Templates FOLDER contains links to

templates. Strictly speaking, we can present folder objects in a template form. However, the point of have folders is to allow collections of objects to be viewed in special ways. Therefore, folder objects are shown differently than others. There are two ways of showing the links in a folder. One is to use a network, as in the case of the User Templates folder. Another is to display the links in a tabular fashion. An example of this will be shown later. We now return to how a user can create objects from the templates.

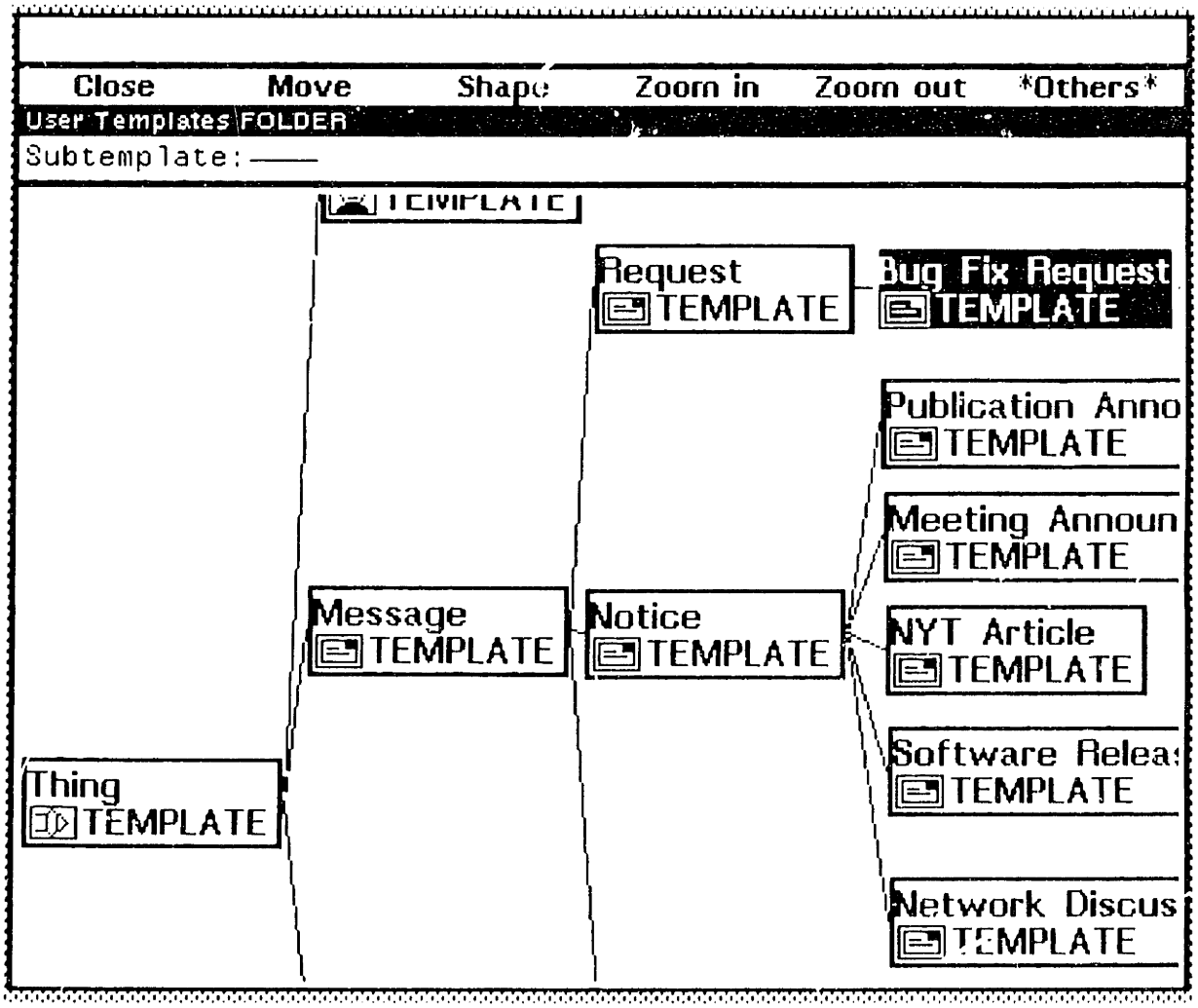


Figure 1: The User Templates FOLDER contains all the templates of the system. The BUG FIX REQUEST template is shown to be selected. The "Close" action closes the window. "Move" moves it to a position specified by the user. "Shape" allows the user to reshape the object. "Zoom in" shows a closer view of the network (*i.e.* spreads out the links). "Zoom out" does the opposite – pulls the links closer together. "*Others*" gives up pop-up menu of other actions, which we will not describe here.

To create an object, the user first selects a template by pointing the mouse at its link in the User Templates folder and clicking the left mouse. Then she clicks the "action" mouse button (currently, this is the right button) with the mouse within the folder to bring up a menu of actions that can be applied to templates. One of these actions is Create Object. Suppose the user chooses this on the BUG FIX REQUEST template. A BUG FIX REQUEST object is then displayed on the screen.

Figure 2 shows a BUG FIX REQUEST object with the default values in the fields. The default value for the Bug: field is an object, as is shown by the link to the my bug BUG object. The user can edit the value(s) of any field by deleting and typing in strings. An easier way is to use the aids provided by Object Lens. Figure ? shows what happens when the user first clicks the mouse over the To: field, and then ove the Alternatives option. Choosing Alternatives pops up the side menu which contains the set of alternatives that appropriate for the To: field. The user select one or more of these alternatives. Selected alternatives are automatically put into the field.

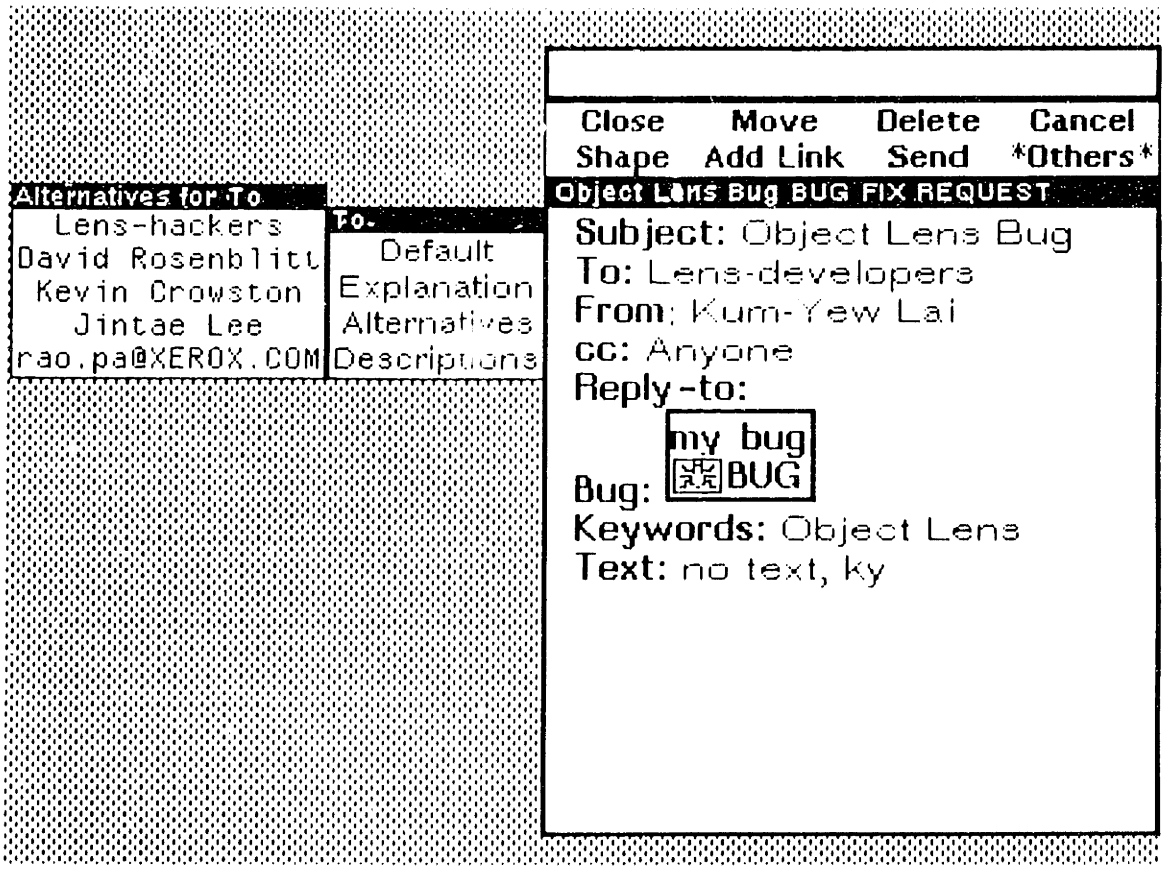


Figure 2: A BUG FIX REQUEST object. Most of the actions shown in the action menu have been described in the caption to the previous

figure. "Delete" deletes the object. "Cancel" is like close, except that the any changes made during this most recent editing session are undone. "Send" sends a copy of this object to the addresses; the original object is closed. "Add Link" is described in the text.

LINKS TO OBJECTS PROVIDE REFERENCES CAPABILTIES AND FACILITATES BROWSING

The use of links is already familiar to many people. The scientific literature makes extensive use of them in the form of citations of the literature. Authors and supervisors put various kinds of comments (*e.g.* links to ARGUMENT objects) in the margins of drafts. Secretaries and librarians use cross references in folders or catalog cards when they want to classify documents into one categories. In Object Lens, we explicitly support the use of links.

In the previous figure, we show a link to a BUG object. The user can see this object by clicking the "action" mouse to get a pop-up menu of actions that are applicable to BUG objects. Choosing Show displays the object on the screen; Figure 3.

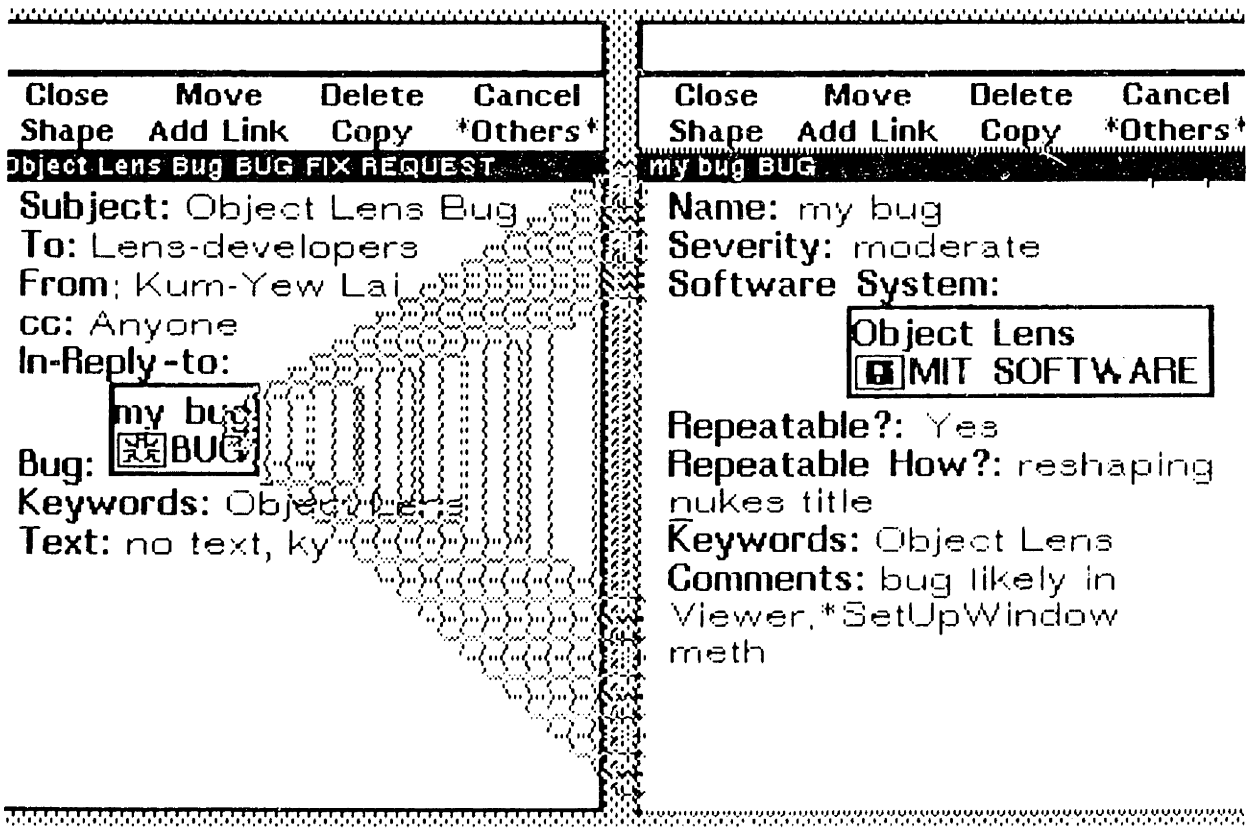


Figure 3: A BUG object has a link embedded in a BUG FIX REQUEST object. The strips of vertical lines represent a transient (it is shown here for illustrative purposes), animated trail showing that the BUG link refers to the BUG object.

Links to the same object can be placed at different places (*e.g.* some onthe screen, some in other objects). Therefore, they serve as references to the underlying object; copies of the object need not be made. Besides the benedfits of economy (fewer objects need to be created) and efficiency (there is no need to keep track of copies so that changes to one propagates to the others), this scheme also allows changes to the object to be seen regardless of which link the user uses to show the object. Suppose, in Figure ?, the user changes object C by clicking the "action" button on its link on the screen, editing it, and closing it (bringing it back to limbo). If she subsequently Shows the object again, whether from the link on the screen or the link in object B, then the previous changes would be shown. Also, if the object is already on the screen (*e.g.* the user brought object C up using the link on the screen), then choosing Show again on a link (whether the same link on the screen, or the link in object B) will cause Object Lens to show the animated trail shown in Figure 4 from the second link to the displayed object.

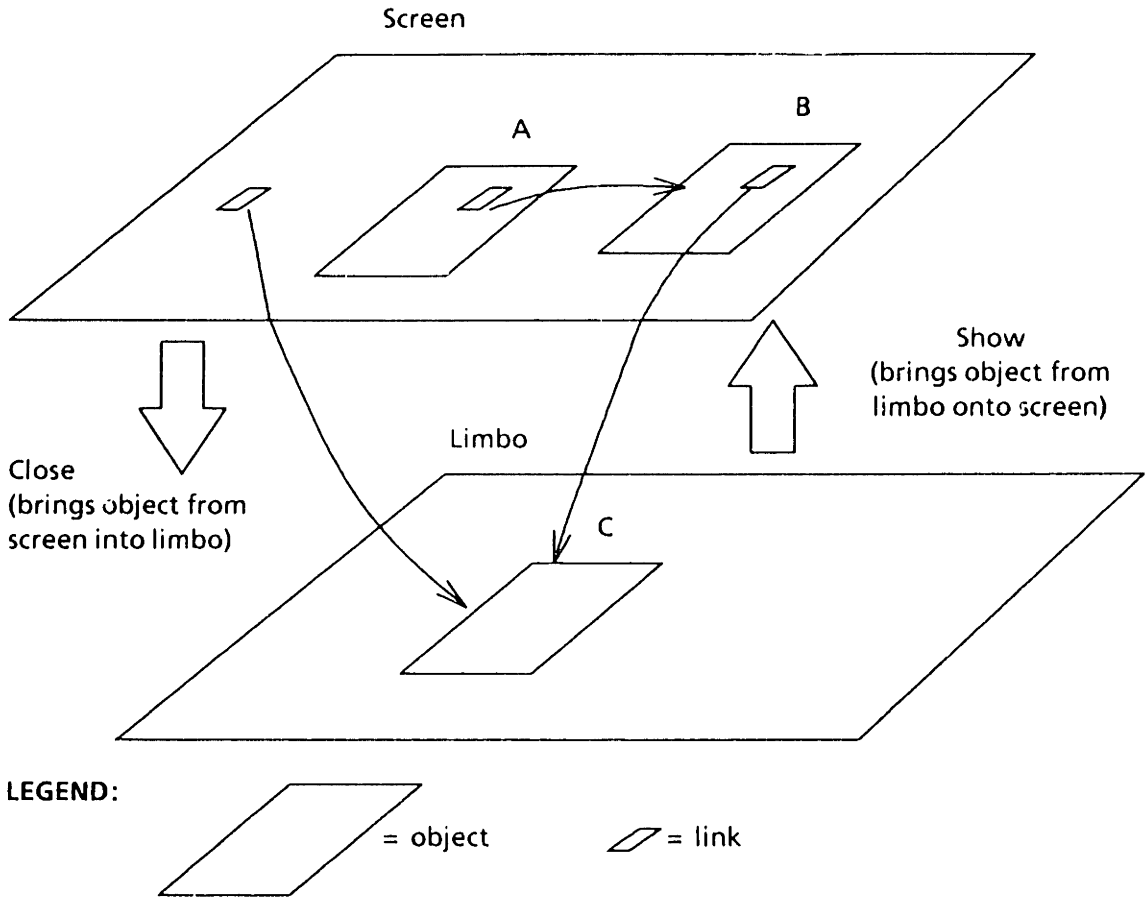
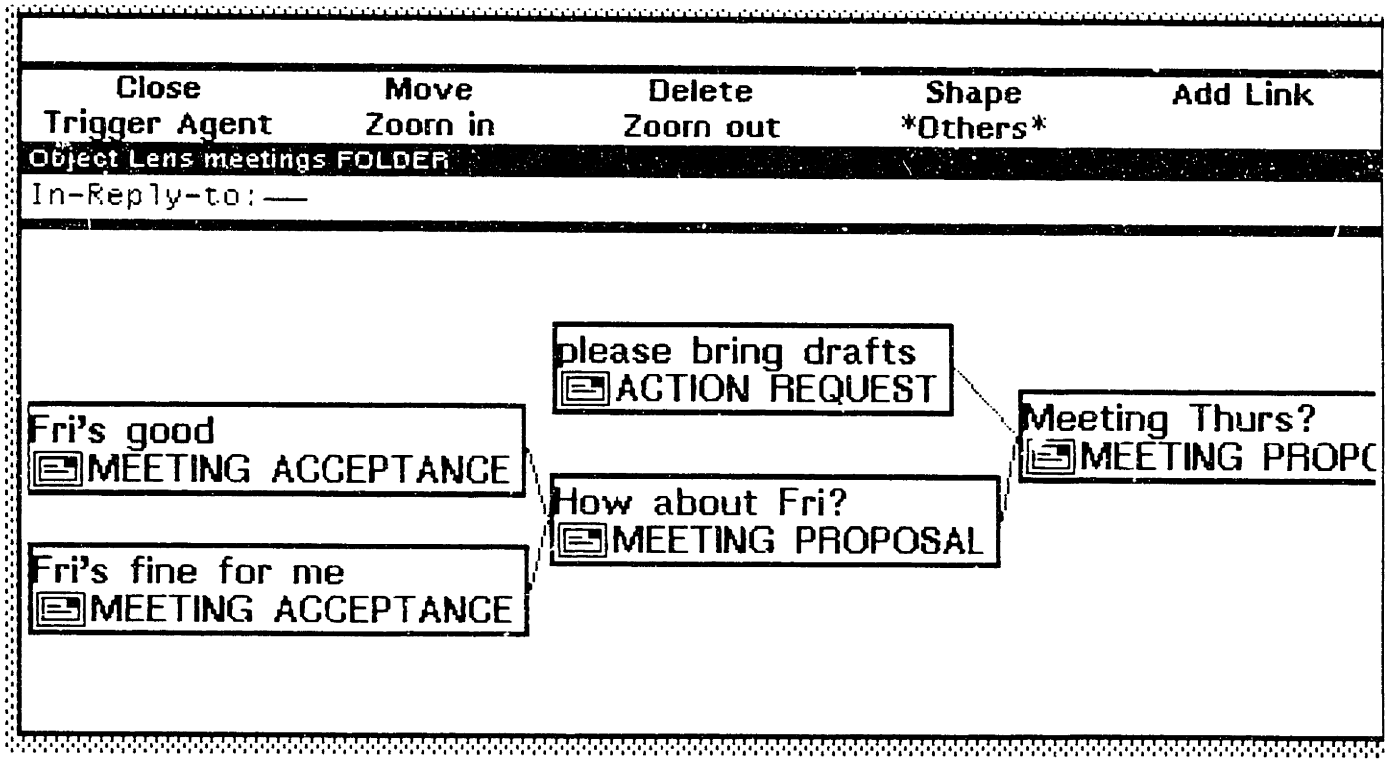


Figure 4: The Show action on a link brings the object onto the screen. The Close action does the converse. Note that objects need not have any links at all (e.g. object B). Also, object C has two links: one on the screen and one embedded in object B.

Besides being used for references to objects, links also form the basis of a hypertext system. A user could construct and store a network of objects with embedded links pointing to other objects. To put links into an object, she first chooses the point in the object where the link(s) should, and then choose the Add Link action. This allows her to use the mouse to select objects whose links would be added into the object. To look at a chain of objects connected by embedded links, the user can look at one object, follow (*i.e.* use the Show action) the embedded link to its object, and in turn, follow an embedded link in the second object; and so on. If the objects are all in a folder, then the user can also display them as a network. Figure 5 shows a network of message objects being connected using the In-Reply-to: field.


Close
Trigger Agent
Move
Zoom in
Delete
Zoom out
Shape
Others
Add Link
Object Lens meetings FOLDER
In-Reply-to:
please bring drafts
ACTION REQUEST
Meeting Thurs?
MEETING PROPC
Fri's good
MEETING ACCEPTANCE
How about Fri?
MEETING PROPOSAL
Fri's fine for me
MEETING ACCEPTANCE

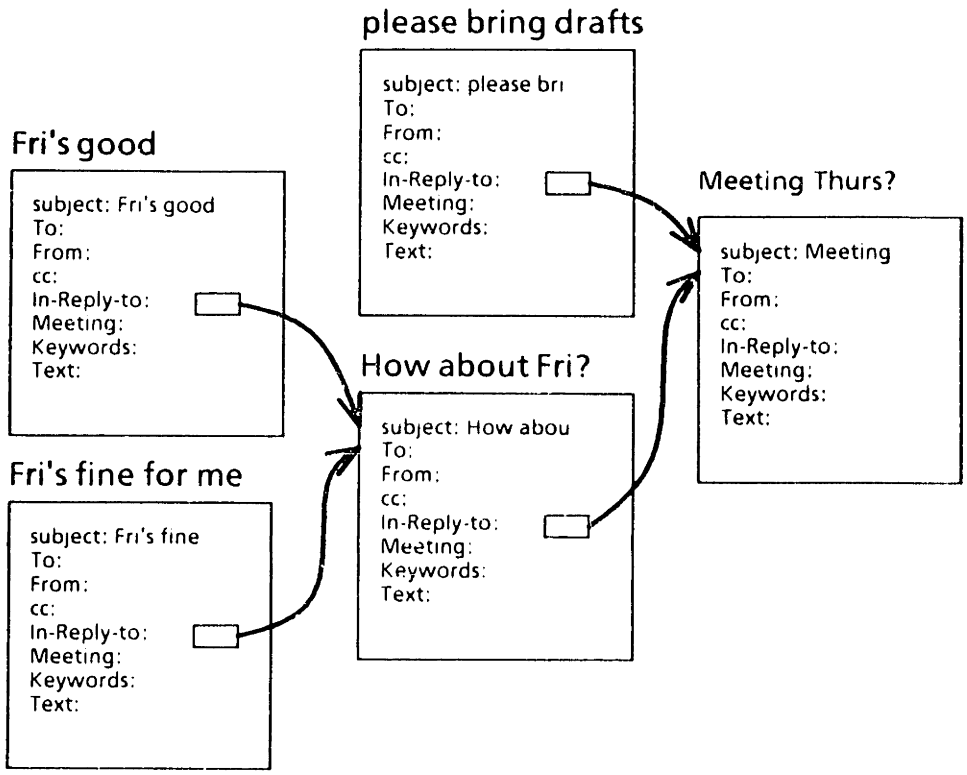


Figure 5: The Object Lens meetings FOLDER. This folder shows the Reply-to: fields connecting the links in the folder.

In a subsequent section on the interface, we describe how the user can create folder networks like the one shown above.

USING DESCRIPTIONS TO DEFINE ARBITRARY SETS OF OBJECTS

As mentioned earlier, the side menu that pops up at the side of an object has an option called Descriptions. If the user chooses this, a network of templates would be shown; Figure 6.

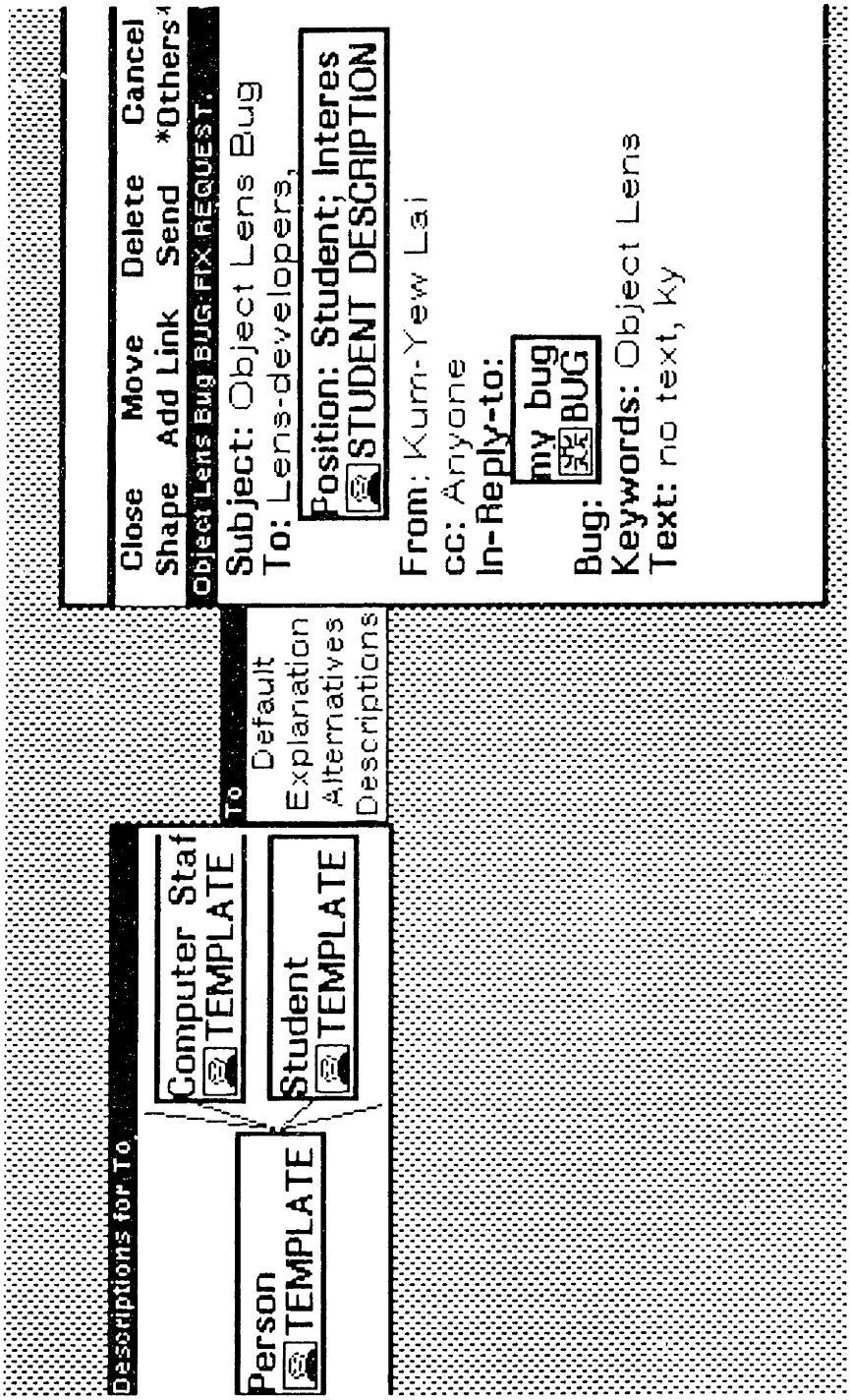


Figure 6: Choosing the Descriptions option in the side menu pops up a network of templates. The user can use this network to add descriptions into the field.

The use of the network of templates is similar to that of the list of alternatives that shows on choosing the Alternatives option. But instead of adding a string into the field, Object Lens adds a description. In the figure, the user has chosen the STUDENT template. The description is shown in Figure 7.

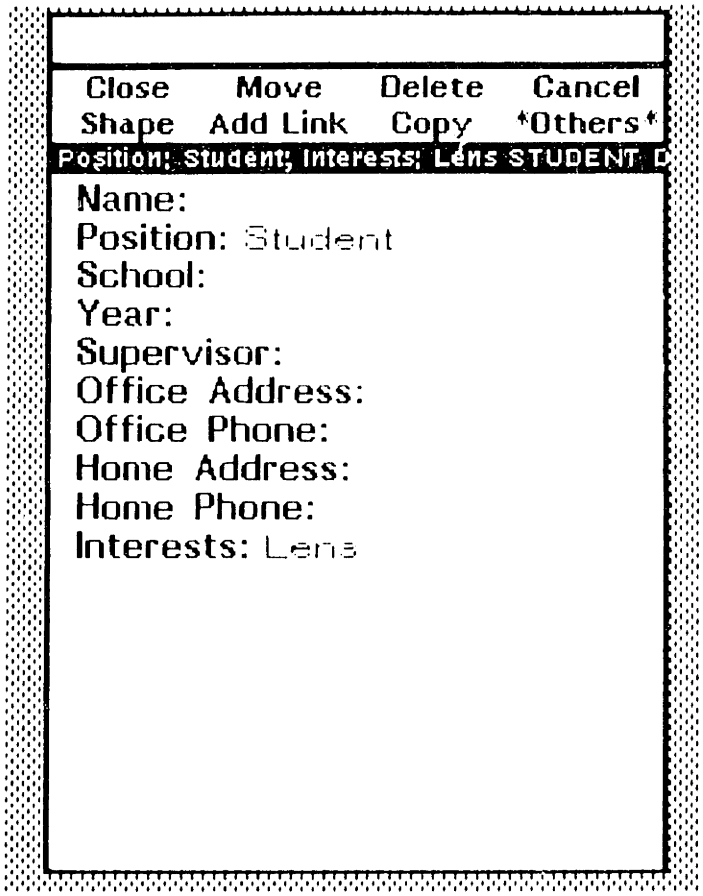


Figure 7: The STUDENT DESCRIPTION shown in the previous figure.

The description has the same fields as the STUDENT template. Therefore, the description in the example represents the set of all STUDENT objects which have the value of "Student" in their Position: fields (incidentally, this is a default value filled in by the system) and the value "Lens" in Interests:.

If the user were to choose the Send action on the BUG FIX REQUEST, the object will be sent to the addressees. The description will be resolved into a list of objects, whose names (*i.e.* strings) would be added as values into the To: field. Therefore, the user need not know explicitly who are the students interested in Lens.

AGENTS AND RULES DEFINE THE BEHAVIOR OF THE SYSTEM

The main use of descriptions is in the RULE objects. Rules are collected into a folder, which is in turn used as the value of the Rule Folder: field of an agent; Figure 8.

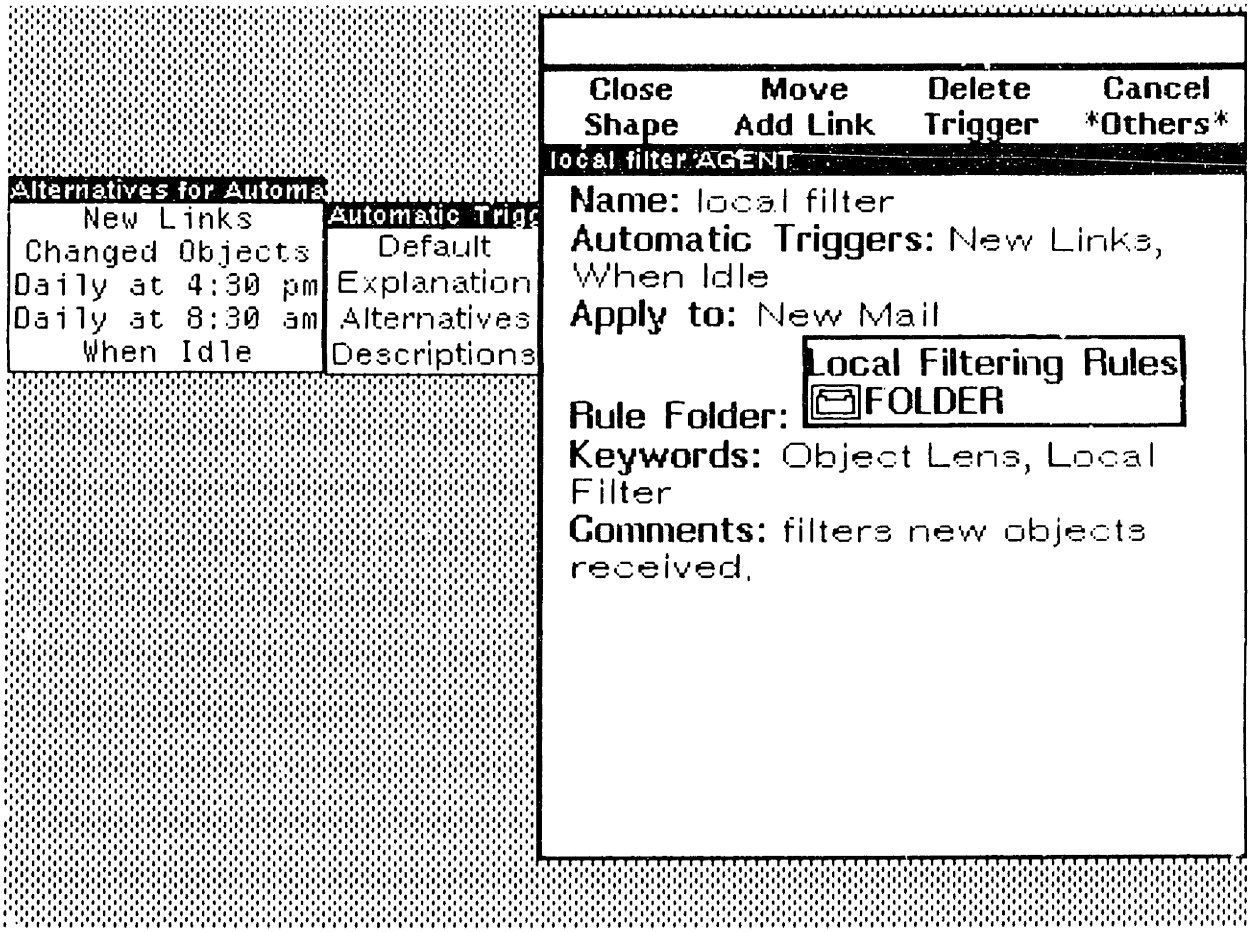


Figure 8: An agent which acts on objects in the New Mail FOLDER. Notice that one could put the link to the New Mail FOLDER in the Apply to: field. Since that field expects a folder object, the user can put in a string too. Strings are treated as objects when the field expects to get objects and not strings.

The Automatic Triggers: field contais information about when and how often the agent object should run its rules. The Apply to: field is a collection of folders. If the agent is triggered, then it runs its rules (in the folder in its Rule Folder: field) over the objects in the folders in the Apply to: field.

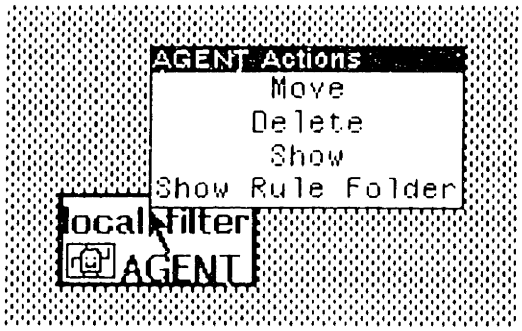


Figure 9: Clicking the "action" button on a link to an agent object gives a menu of applicable actions. For most objects, only the first three are shown (other appicable actions are shown in the top menu of the object when it is shown – this is to avoid having too many actions in the pop-up menu).

While it is possible to call the Show action on the rule folder of the agent, it is common for users to want to see the rule folder without showing the agent object. Figure 9 shows the action menu for the link when it is on the screen. The Show Rule Folder shows the rule folder directly (the Show action shows the agent object). Figure 10 shows the Local Filtering Rules FOLDER object. Incidentally, we show it in tabular form (as opposed to the network form of the User Templates FOLDER shown earlier).

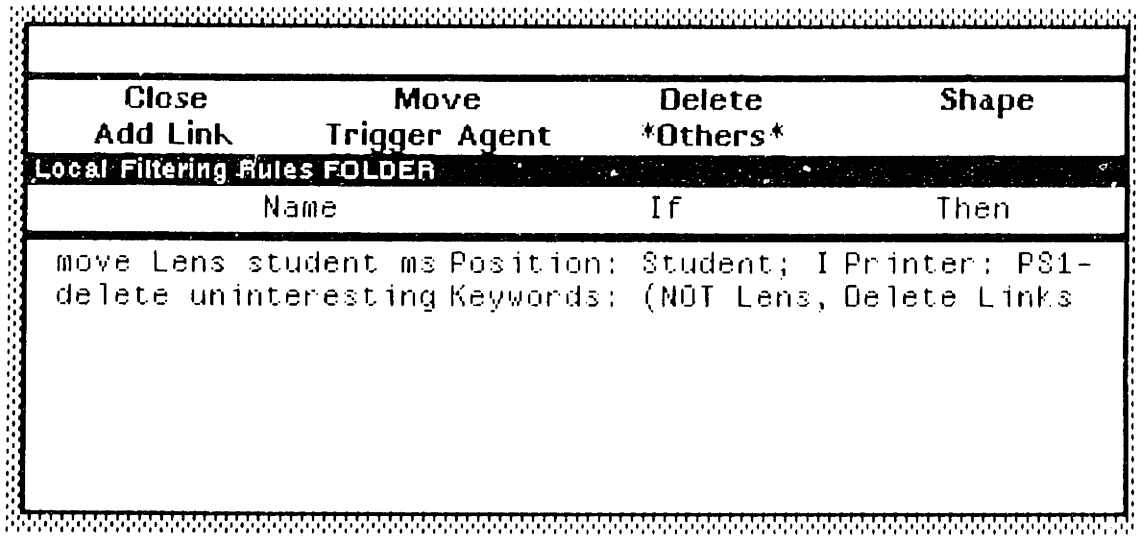


Figure 10: The Local Fitlering Rules FOLDER. All FOLDER objects have a Trigger Agent action. Choosing this will pop up a menu of possible agent s(*i.e.* those which have the folder as one of the values in their Apply to: field). Add Link adds a link into the FOLDER, just like in ordinary objects shown as forms.

Figure 11 shows a RULE which uses the description discussed earlier. The user can use the Add Link action of the RULE object and select the STUDENT DESCRIPTION in the BUG FIX REQUEST of the earlier example.

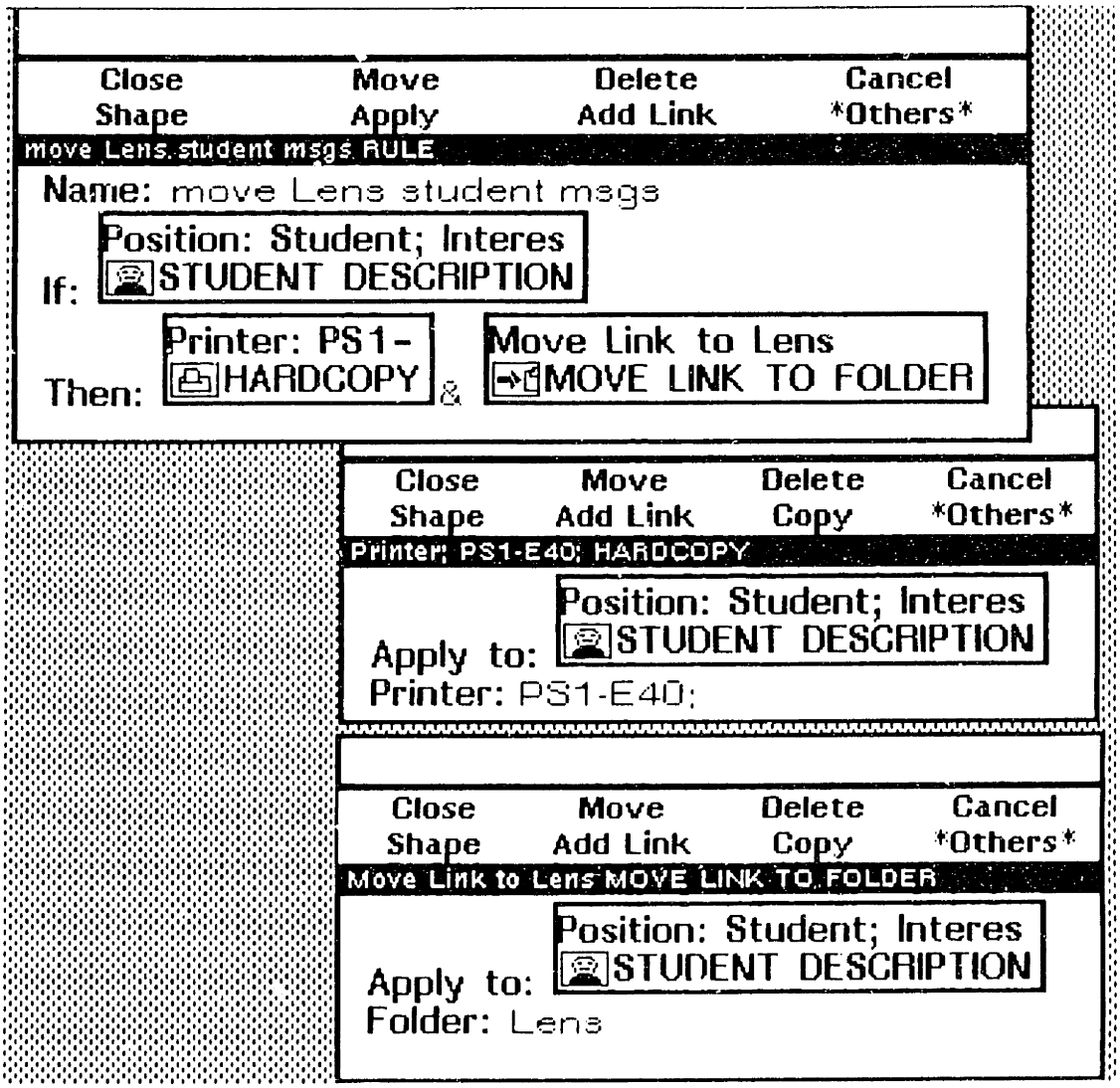


Figure 11: A RULE object. The If: field is filled with a link to the STUDENT DESCRIPTION earlier. There are two ACTION objects are shown as separate objects.

There are two actions in the Then: field. One hardcopies the object which satisfies the If: field (as indicated by the Apply to: field – the STUDENT DESCRIPTION is used as a variable and the scope is the RULE object), and the other moves the link from the New Mail FOLDER to the Lens FOLDER.

Some interesting actions for RULE objects include:

- Trigger: triggers the agent specified in the Agent: field of the ACTION.

- Change Template: changes the template type of an object, for example, from a BUG FIX REQUEST to a LENS BUG FIX REQUEST if "Lens" is one of the keywords.

The goal of collecting rules into agents is to modularize the kind of activities and work that needs to be done. For instance, the local filter AGENT deals with incoming mail. We can also have agents that look at, say, a folder containing links to MEETING objects. It fires at 8:30 every mornig and reminds the user of her meetings..

It is cumbersome for the user to apply the Show action on the two action objects so that she can edit them if she wants to. The next section shows how one can customize the interface to have the ACTION objects shown automatically. Also, there are some fields that are not shown in the RULE and ACTION objects, such as Name: (for the ACTIONs), Keywords:, and Comments:. Again, the showing of fields is customizable.

CONSISTENT AND CUSTOMIZABLE INTERFACES

In many cases, users want customize their views of the objects world. Each field in a template has the following five aspects: Customized Name, Show Link Object, Show in Forms, Show in Tables, and Show in Networks. These aspects control the presentation of objects on the screen. To change these aspects, the user needs to click the "action" mouse button on a template link, and choose the Change Display Format action. This presents a pop-up menu of the names of these five aspects. The user chooses one of these, and an aspect editor will be presented. The five aspects works as follows:

- Users can rename fields provided in the templates by changing the Customized Name aspect. For instance, the aspect editor for the MESSAGE template is shown in Figure 12.

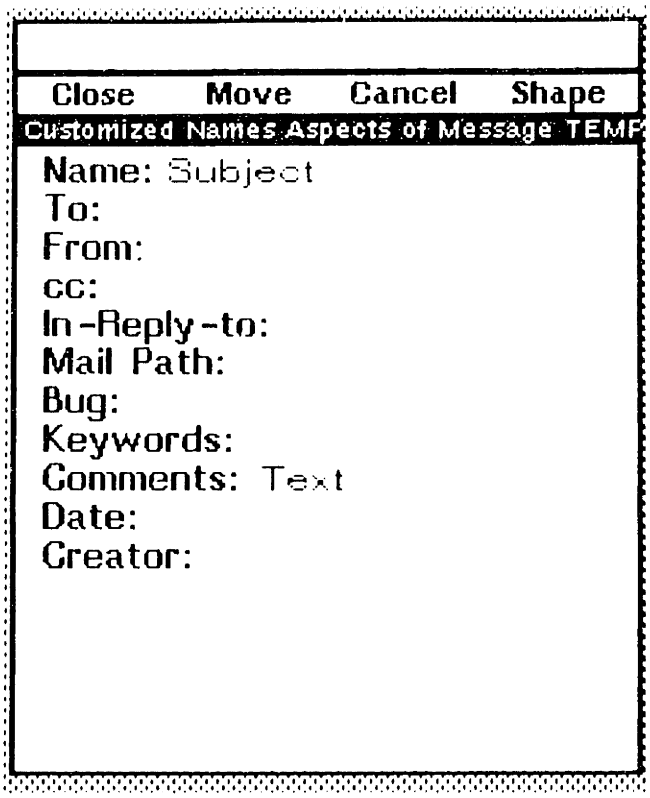


Figure 12: The Customized Names aspects of the MESSAGE template.

Therefore, whenever a message object is displayed, its Name: field is shown as Subject: and its Comments: field shown as Text:.

• User can control whether links in the values of the fields of an object should be show their respetive objects. If the aspect for a field has the value "Yes", then any link in that field will get the Show action – *i.e.* its object will be displayed (currently, the object will be displayed in a region to the right of the embedding object). In the RULE example in the previous section, the user can have the ACTION objects shown automatically, when the RULE is shown, or when the links to the ACTION objects are added into the RULE object. This is done with the Show Link Object aspect of the RULE template changed as in Figure 13.

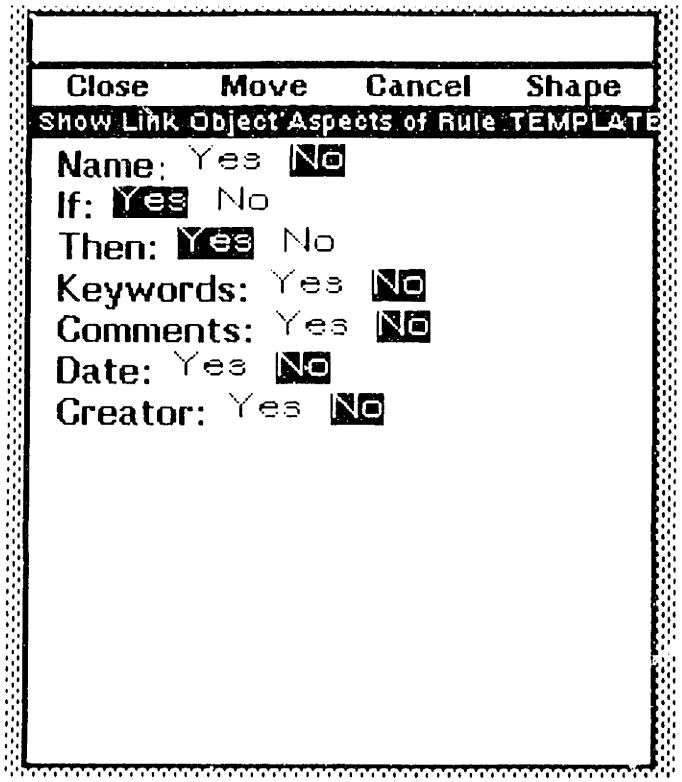


Figure 13: The Show Link Object aspect of the RULE template. Links in the If: and Then: fields automatically show their objects when they first get into the either field (*e.g.* the user adds an embedded object into either field), or when a RULE object with embedded objects in either field is displayed.

• Fields can be shown or hidden by the user. This is done using the Show in Forms aspect of the fields. Figure 14 shows how the user can suppress the Mail Path:, Date:, and Creator: fields when BUG FIX REQUEST objects are shown.

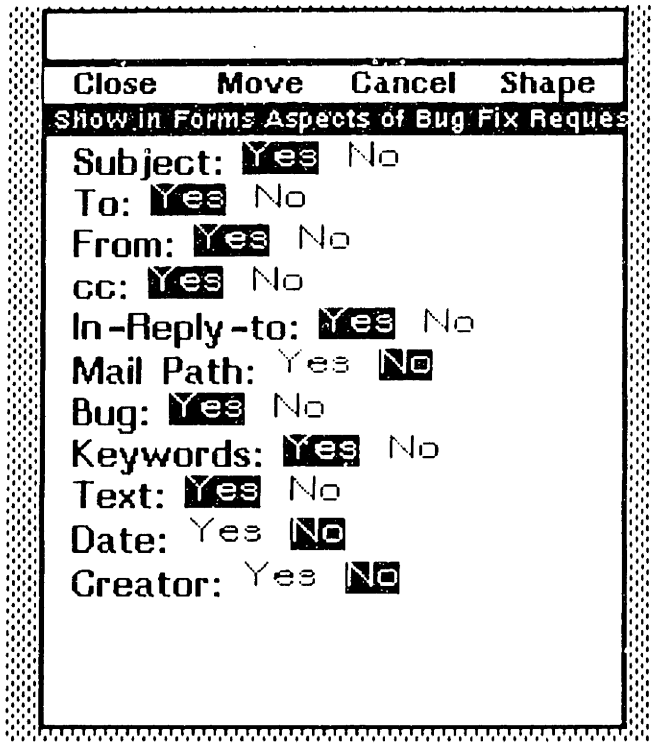


Figure 14: The Show in Forms aspect of the BUG FIX REQUEST template. The Mail Path:, Date:, and Creator: fields are not shown. The user can also change this aspect at the object level (*i.e.* change the aspect for a particular BUG FIX REQUEST, not to the template).

• For a folder whose links as displayed in a table, the fields displayed can also be customized. This is controlled by the Show in Tables aspect of the fields. For instance, the Local Filtering Rules FOLDER shown earlier has the aspect editor shown in Figure 15.

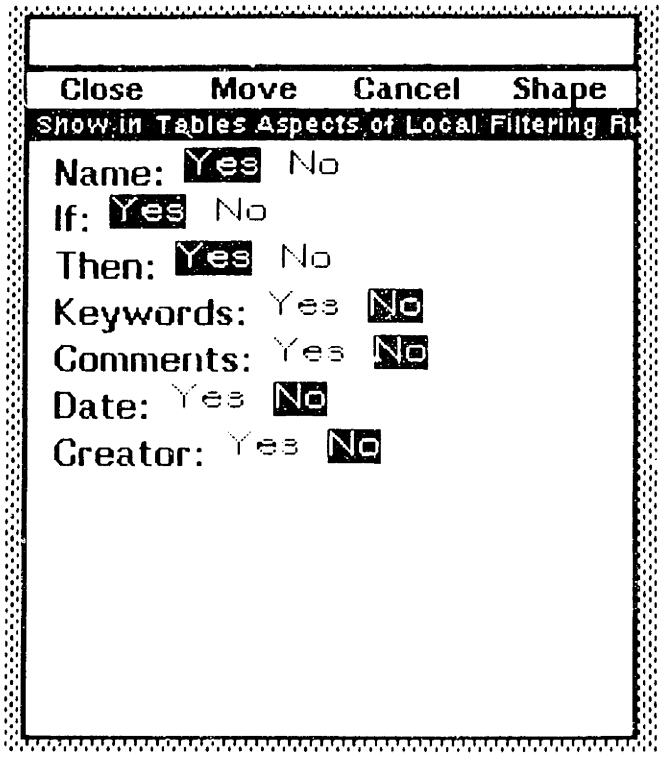


Figure 15: The aspect editor that can change what fields to show in the Local Filtering Rules FOLDER.

While the previous aspects are changed at the template level, the Show in Tables aspects can be changed for individual folders as well. That is, if the Show in Tables aspect editor shown here is actually the one for the RULE template, then all folders collecting links to RULE objects will display, initially, the Name:, If:, and the Then: fields. Of course, the user can change the format for the individual folders after that.

• Just as the fields displayed in a folder shown as a table can be changed, the fields displayed in a folder shown as a network can also be changed. For instance, the Object Lens meetings folder has a Show in Networks editor which has "Yes" for only the In-Reply-to: field, the user can change it so that the message objects are linked by another field. If more than one field is shown, then differnt edges are used (*e.g.* some with dashes). In its current implementation, the maximum number of fields that can be shown is six.

Finally, we will not dwell into the issues of graceful degradation and incremental adoption of the system. These are sufficiently discussed in the beginning.

FUTURE DIRECTIONS

- Agent-oriented programming.

Agents as they are currently constituted are pathetically inadequate. They do not form plans for actions, they can delegate tasks (using the Trigger action to trigger other agents) but do not have the ability to handle result reporting, they do not know how to handle exceptions, and they are not resource conscious.

One intriguing research direction is the formulation of agent-oriented programming as a basis for supporting cooperative work.

- Integration of the Anyone Server with Object Lens, and generally, remote agents.

The Lens system has a public mailbox named "Anyone". It contains the pool of messages sent by users for redistribution. Receivers who want messages from this pool write rules to retrieve them. An extension of Object Lens with this capability may take the following form.

The Apply to: field of agents can take folders on remote workstations. The general format for indicating a remote folder can follow well-established conventions in mail systems. For example, it may be of the following syntax:

<folder name>@<mail pathname>.

Therefore, if a user wants to have her agent run its rules over objects in the New Mail folder of the Anyone mailbox, then the value of the Apply to: field of the agent is "New Mail@Anyone". Similarly, if she wants to run her agent on

A problem with using explicit folder names in the Apply to: field is that the user must know in advance the new of the folder containing the objects of interest. This problem becomes exacerbated with remote folders. One solution is to establish conventions for public accessible folders (*e.g.* the objects that others can run their rules on are all in the folder called Public). This is limiting since it depends on conventions within a group; but this can be alleviated by putting the names of publicly accessible folders into the Alternatives aspect of the Apply to: field. A better solution should involve, at the minimum, the client specifying what

objects to run and the owner of the objects specifying what objects should be made public.

After the user has created the agent, Object Lens would automatically send the agent to the owner of the folder in the Apply to: field. This could be done using the existing REQUEST FOR INFORMATION template, with the agent as the value of an additional field called How?: (*i.e.* how should the information requested should be retrieved). Normally, the client specifies how the information might be retrieved; but the agent would be just as good a value, since it can be interpreted as "retrieve all the objects that my [the client] agent wants."

The receiver of the REQUEST FOR INFORMATION can act on the agent received. Just like she can execute the Load File ACTION on a BUG FIX ANNOUNCEMENT (which loads the file specified in the Patch Fiel: field of the announcement), she can also execute the Install Agent ACTION on the REQUEST FOR INFORMATION (which installs the agent specified in the How?: field of the request). Installing an agent means making it receptive to the events specified in its Automatic Triggers: field, as well as to the user (*e.g.* when she issues the Trigger Agent ACTION) or other agents (*e.g.* they execute the Trigger ACTION).

The simple scheme above can form the basis of the Anyone server. It would have a rule that automatically installs all foreign agents (agents belonging to other users). Of course, the rule should check that an agent has detrimental side effects like deleting the objects in the Anyone server.

CONCLUSION: OBJECT LENS CAN FORM THE BASIS OF A GENERALIZED TOOL FOR COOPERATIVE WORK

We have shown how Object Lens can smoothly integrate three formerly isolated technologies. We have also shown the features of a system like Object Lens can support many actiities in cooperative work like information sharing, conversations for action, and the retrieving and browsing of complex information. In addition, systems like Object Lens accumulate objects over time, and can be viewed as a form of informal knowledge engineering.

OBJECT LENS: A DETAILED DOCUMENTATION

ABSTRACT

We describes a more general version of the Lens system (Malone, *et. al.*, 1987) called Object Lens. It extends the idea of semi-structured messages to semi-structured objects. This paper serves as a detailed documentation of all the features of the system. At various points, we also discuss the design alternatives and decisions made. As of this writing, the system as it is described here is only partially implemented. An older version of Object Lens has been more fully implemented; therefore, this paper also documents the conceptual design for the next version of the Object Lens system.

INTRODUCTION

Lens is an intelligent information sharing system that provides the following primary capabilities (Malone, *et. al.*, 1987):

- senders fill in their messages in structured templates; this helps them in composing messages.

- receivers can specify production rules to automatically process incoming messages. The processing could include filtering and classifying messages, as well as executing some actions like the loading of files (*e.g.* in bug fix announcements).

- senders can send their messages to a public mailbox called "Anyone" to redistribute the message to interested users.

- users can specify production rules to automatically retrieve messages from the "Anyone" mailbox. Therefore, they can get messages that satisfy their criteria, and which they might not otherwise have seen.

We describe a more general version of the Lens system. The following key ideas form the basis of the Object Lens system:

• it extends the idea of semi-structured messages to semi-structured objects. This extension increases the power of the Lens system by unifying messages with other objects to form a knowledge base. In general, the increase in power come from the following capabilities:

• the retrieval of information. For instance, this helps senders in composing messages (*e.g.* a sender can retrieve those PERSON objects that are in the Painting Division, and put links to these objects in the message to send).

• the browsing of information. Using the above example again, the sender can browse the information stored in her workstation

• the use of abstractions. For instance, a sender only need to know what kind of people (*e.g.* those in the Painting Division) she wants to send her message, and not exactly who might satisfy that description (*e.g.* John Brown, Peter Green, *etc.*).

• enhanced automatic processing of information. A receiver may update her knowledge base by processing her incoming mail. Conversely, she may process her incoming mail based on the information already existing in her knowledge base.

• the modularizaton of information. Objects introduce modularity in information. This leads to economy of specification (for example, one need to specify just a BUG template with BUG FIX ANNOUNCEMENT and BUG FIX REQUEST templates embedding the BUG template. It also leads to a more flexible interface. For instance, a sender can put a link in her BUG FIX ANNOUNCEMENT message, with the link pointing to the BUG object embedded in an existing BUG FIX REQUEST.

These increased capabilities span across various functions – in constructing messages and other objects, in automatically processing incoming messages and other objects, and in the presentation of information.

• it exploits the synergy of combining various existing technologies like hypertext systems, knowledge based systems, and coordination tools. This synergy increases the power of any of these technologies existing in isolation. Hypertext systems introduce an informal representation for knowledge based systems and enhances the connectivity between objects in coordination tools. Knowledge based systems introduces processing power to hypertext and coordination tools. Finally,

coordination tools allow distributed and shared hypertext and knowledge based systems.

• the use of links pointing at an object introduces the notion of an abstract reference. With such an ability, one can deal with many manifestations of a single object on the screen. For instance, two folders may each have a link pointing to the same object. This is a fundamental difference from systems like Lafite, where copies of objects have to be made in the two folders. Therefore, changes to one copy (*e.g.* putting a "seen" mark to indicate that the user has seen the object) does not propagate to any other copy. Also, the use of links allow the use of links as variables. In an example below, we see that RULEs can have their actions refer to whatever that satisfies their predicates.

IMPLEMENTATION OVERVIEW

The system is written in InterLisp, using the Loops, an object-oriented programming tool. It runs on Xerox 1100-series machines connected by Ethernet networks. Mail services are provided by the Xerox Network Services (XNS) protocol. As of this writing, the system as described is only partially implemented. An older version of Object Lens has been more fully implemented; this paper also documents the conceptual design for the next version of the Object Lens system.

OBJECTS AND TEMPLATES ARE THE MAIN ENTITIES IN OBJECT LENS

The main entities in the system are *objects*. Each object is like a form; it can have one or more fields, and each field can have any number of values. Figure 1 shows a Bug object.

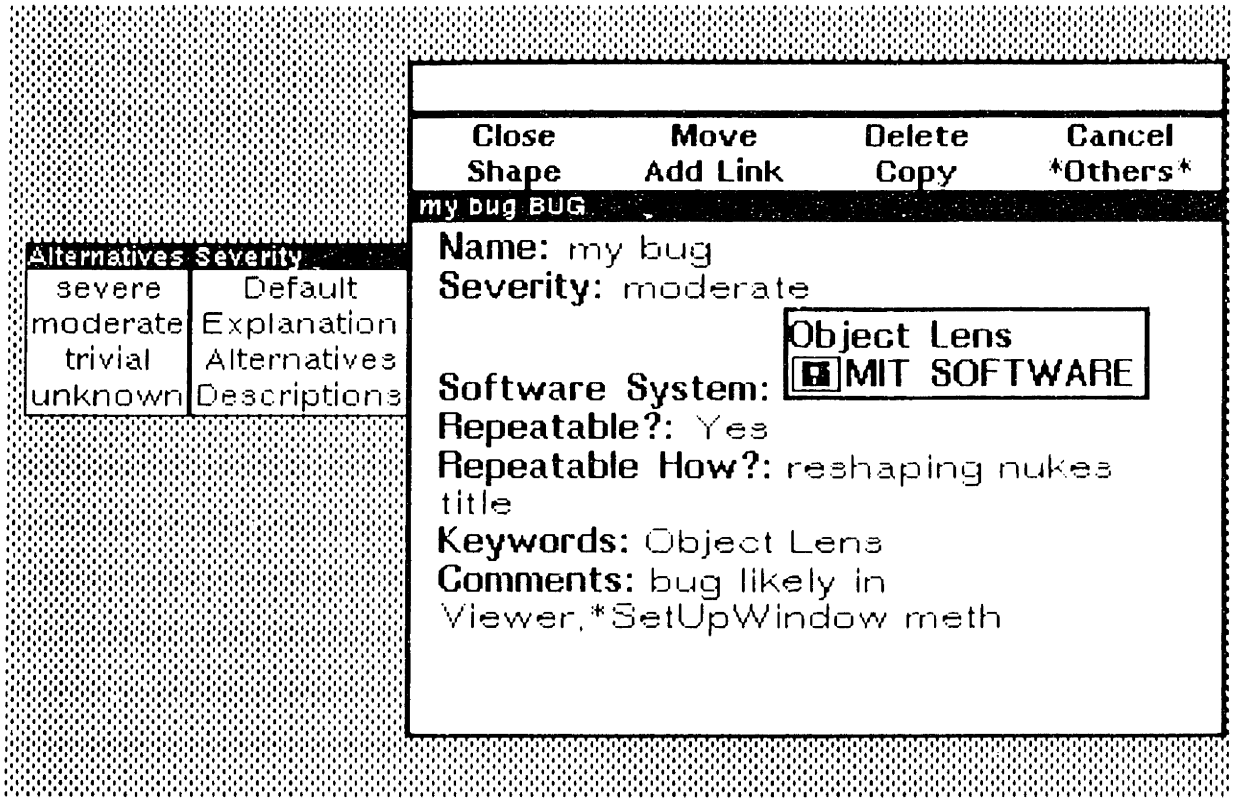


Figure 1: A Bug object. The Close action closes the displayed object. Move moves it a user-selected position on the screen. Delete deletes the object, after the user confirms. Cancel is like close, except the lastest changes are not incorporated into the object Shape reshapes the object window. Add Link gets links selected by the user into the object at the current cursor position. Copy makes a copy of the object on the screen. Clicking on *Others* pops up a menu of actions that are less frequently used. These include Shrink (turns object into link), Hardcopy (prints object using default printer; embedded objects printed as shown on screen), Change Template (change from BUG template to some other template), Change Display Format (please see section on the presentation of objects), Grab Link (like Add Link, except that it also deletes the original link) and Show History (prints the rules that have fired on this object). The title of the object shown consists of the a concatenation of its name (*i.e.* my bug) and its template name (*i.e.* BUG).

The values of a field may be text strings or links to other objects. In Figure ?, the Software System: field has a link to an MIT Software object called Object Lens. An object can have any number of links. Some of these may be embedded in other objects, while others may be on the screen; please see Figure 2.

Figure 2: A link to the my bug object in Figure ?. The top line in the link is the value of the Name: field. The icon at the bottom left is the icon for the BUG template. The bottom line is the template name. Clicking the mouse with the right button on the link produces the popped up action menu as shown. Move moves the link to a user-selected position on the screen. Delete delets the link. Show displays the object of this link.

The objects are created using *templates*. Templates can be thought of as notepads – the user can "tear off a sheet" from the pad to create an object (the sheet). Templates define the structure and behavior of their objects.

Each template has several aspects, as shown in Figure 3. This means that every field in the template can have different values for the different aspects. Three of these aspects influence the values of the object of the template type – these are Default (*i.e.* the value of the template is the default for the object created using that template), Alternatives (these are possible alternative values for a given field of the object), and Descriptions (these are possible descriptions – described later – which can be values of a given field). The other aspects influence how the object of the template type can be displayed on the screen. This structure of a template is passed on to any object of the template type. Objects of a certain template type gets the same fields as the template, and the same aspects (notice that the value of a field in the template serves as the default for the same field in the object).

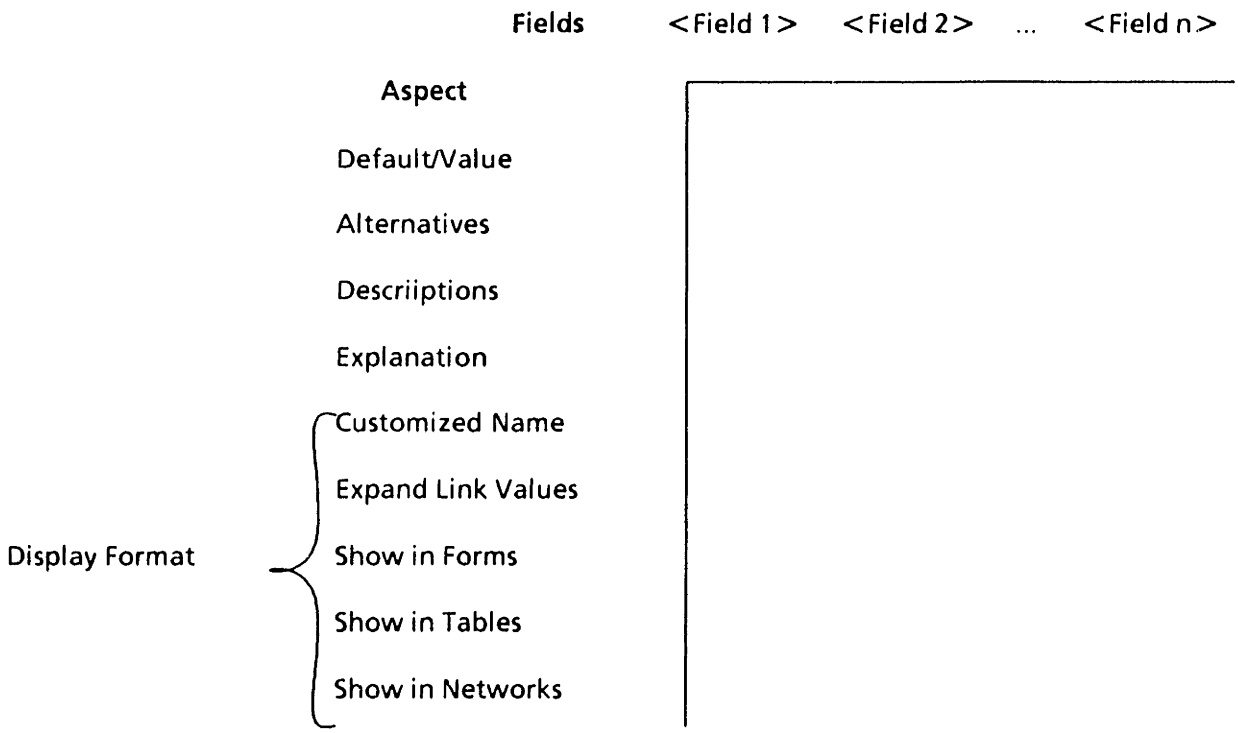


Figure 3: Each object has four aspects. Each field of the object has its own value for each of the four aspects.

Each template has also a list of actions applicable to objects of its template type. Like the template structure, the template actions are passed onto the object of the template type. Therefore, the template determines what the object can do (*i.e.* its behavior).

While the objects of a template inherit various aspects and actions from the template, the templates themselves are organized into an union inheritance hierarchy. The top-level template is called THING. It has subtemplates like MESSAGE, PERSON, and FOLDER. By "union inheritance hierarchy," we mean that the fields of a subtemplate is the union of the fields of its ancestral templates – all the way to the root template THING. Similarly, the values of each field in a subtemplate (except its Name: field) is the union of the values of the same field in the ancestral templates. This is the same for the other two aspects (Alternatives and Descriptions) of the template which are concerned with the values of the template. The other aspects are not union inherited, but override inherited. This means that a field inherits its value from the values of the corresponding field of the ancestral templates only when it does not have any value specified.

Before we continue, a description of FOLDER objects is in order. Like other objects, folders have form structures. However, they have only one field called Contents:. They can be thought of as collections of links of objects. All objects other than folders are displayed as forms. Folder objects have the values of their Contents: displayed as a network or as a table. One can think of this an object whose field (ie. Contents:) is not explicitly displayed, and whose values (of the Contents: field) are displayed in a table or network, instead of being displayed linearly like the values in a form.

NOTATION

In this paper, we will use upper-case letters to print template names and lower-case letter to print object names. The fields of any object are printed with trailing colons. All Object Lens features (object types, object names, fields, actions) are in helvetica font.

SYSTEM OBJECTS

There are a few system objects provided by Object Lens. These provide the new user with a minimum starting point. The system objects are:

- the User Templates FOLDER. This contains the links of templates.

- the New Mail FOLDER. When the user gets her mail, links of the incoming message objects are created and put into this folder. The name of this folder can be changed by the user, like the name of any other object.

- the All Objects FOLDER. This is only virtual, in that the user cannot display it on the screen. It is used for notational purposes – for example, one may want to run rules over all the objects on the workstation, so this folder would be handy (please also see the section on installing agents on remote workstations).

- the Orphans FOLDER. This folder contains the links to objects which do not have links elsewhere, whether on the screen or within other objects.

- the Desk Top LENS OBJECT. The LENS OBJECT template is not a user template – it is not in the User Templates FOLDER. It is the template for some system objects,

like the Desk Top. Figure 4 shows a Desk Top object. The object serves as a top-level control panel for the Object Lens session.

Figure 4: The Desk Top object. Like other objects, the Desk Top objects has a list of applicable actions.

The top line in the Desk Top object reports the status of the user's mail box. This is done through a polling process. Get Mail retrieves message objects from the user's remote mailbox and creates and moves links into the New Mail folder. The presence of the Desk Top on the screen is the criteria for deciding whether Lens is on. Therefore, choosing Close closes the Desk Top object and ends the Lens session. Move is exactly the same Move action in other objects – it moves the object to another position on the screen. Customize puts the Customizer object on the screen (please see below). This is equivalent to the Show action executed on the Customizer object. Save saves the objects in virtual memory into reliable storage. If any the user has created or edited any object since the last time she saved the environment, the Save item is crossed as shown in Figure ?. Objects not created by the user (received through the mail; this is discussed below) are automatically dumped into reliable storage when received. These objects are not user-editable either. Therefore, Save need not be concerned with foreign objects. Saving, if needed, is automatically done when the user ends the session.

- the Customizer LENS OBJECT. This deals with some top-level user parameters.

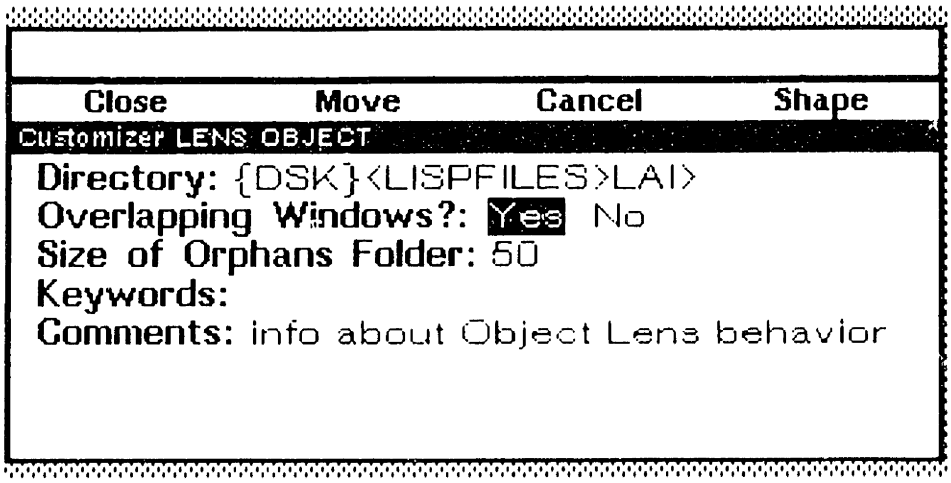


Figure 5: The Customizer object.

Directory: is the directory pathname where the objects are stored. The objects in the system are placed in a region depending on where the Show action is called. This is explained further in the next section. If the value of Overlapping Windows?: is "Yes", then if an object of a given type is placed in a region, subsequent objects of the same type are placed over it, using the same region. If it is "No," then subsequent objects of the same type are placed by the user. To do this, a ghost box denoting the border of the object is attached to the cursor. The user places the box at the destination position and clikcs the left mouse button to anchor a corner of the box. She can then drag the opposite corner to get the desired region, and then click left again to indicate the region desired. The Size of Orphans Folder: field dictates the maximum number of objects that can have links in the folder. A system agent (see subsequent section for more information on agents) deletes the earliest objects to keep the size down. Finally, the Aspect Editor Region: field determines where aspect editors (see below in the section on the presentation of objects) are placed.

CREATING AND EDITING OBJECTS

As mentioned earlier, there is a system folder called User Templates. The user can create an object by "tearing off a sheet" from a template; an alternative way is to Copy from an existing object. We will describe how to create an object using the first method. The second method is described in the section on actions.

The user first selects a template link in the User Templates folder. Following the general philosphy about mouse buttons in selection, clicking the left mouse button on a link within the folder replaces any previous selection with the buttoned link. Selected links are shown in a dark shade. Clicking the middle mouse button adds further links to the selection. The user can hold down the middle button and drag the mouse pointer over several links. This selects all the links in the path of the mouse pointer. Finally, clicking the left mouse button within the folder but not on any link deselects any selected links.

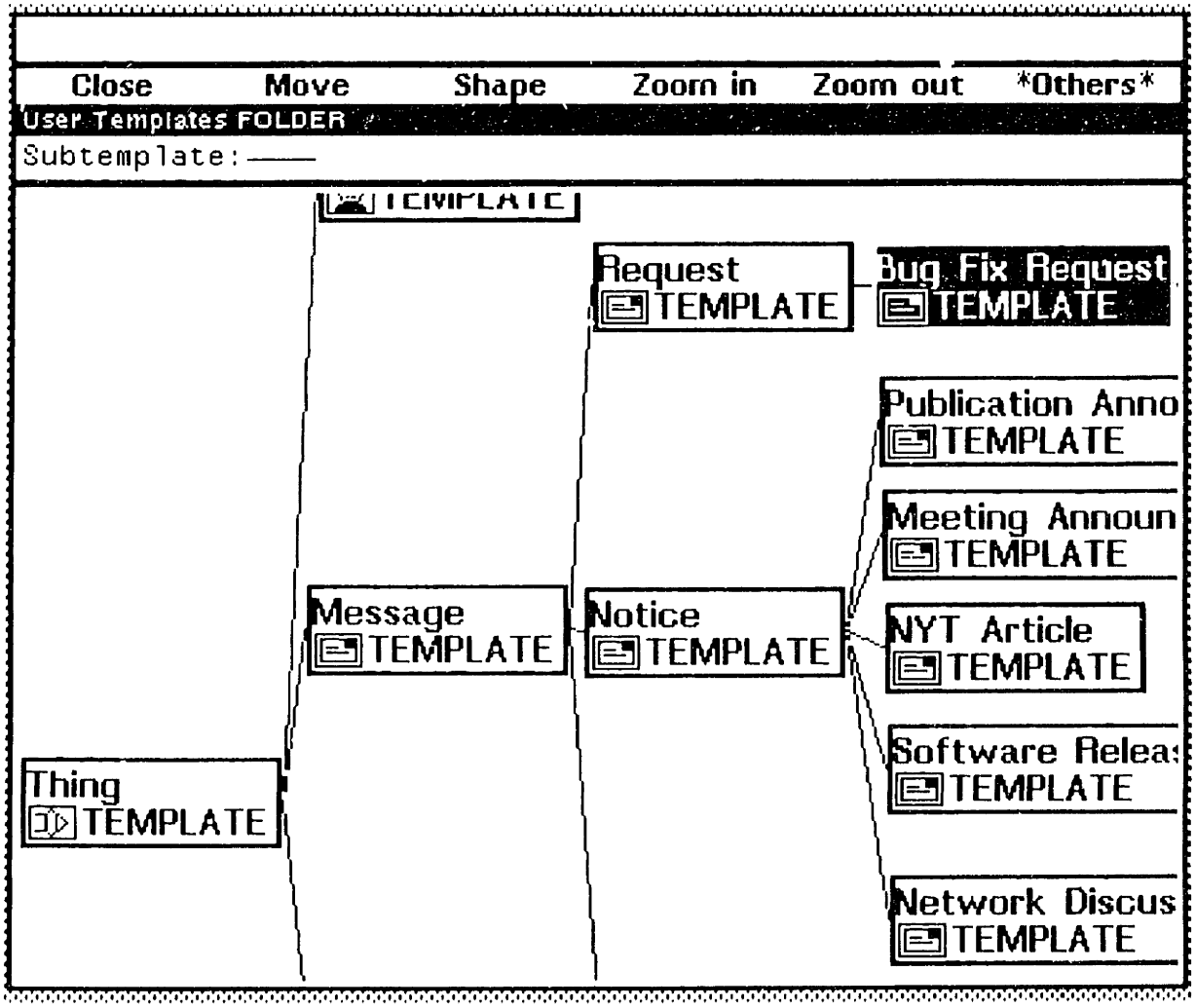


Figure 6: The User Templates folder object. The user has selected the link of the BUG FIX REQUEST template. Zoom in spreads out the links shown. Zoom out makes the links closer to each other. Choosing *Others* gives a pop-up menu of additional actions: Add Object (asks the user for the template type, creates an object of that template, and puts a link to the new object into the folder), Shrink (turns object into a link on the screen), Hardcopy (prints object using default printer; embedded objects printed as shown on screen),

Change Display Format (please see section on the presentation of objects), and Show History (prints the rules that have fired on this object). Besides these actions, there are also other actions of FOLDER objects like Add Link and Trigger Agent.

As in elsewhere on the screen, clicking the right mouse button on the selected links brings up the action menu. If the selection has more than one object link, than the choosen action is executed on each object in the selection.

In this example, we create a BUG FIX REQUEST object. Notice that the BUG FIX REQUEST object, called Object Lens Bug, is initially filled with the values of the BUG FIX REQUEST template. If the value in a field (eg. Bug:) of the template is a link to an object, then the default value put into the new object is a link to a *copy* of the object in the template. In the example here, the BUG FIX REQUEST template has a BUG object in its Bug: field.

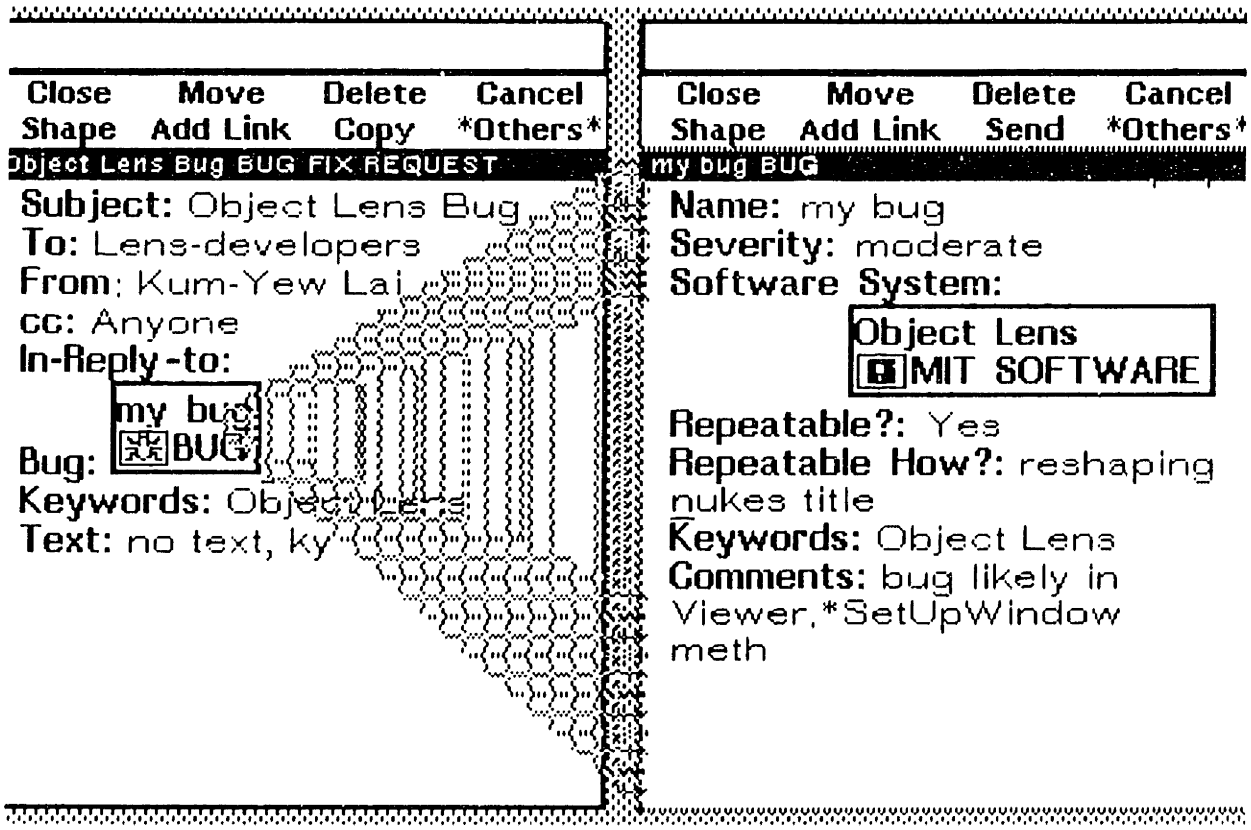


Figure 7: A BUG object has a link embedded in a BUG FIX REQUEST object. The strips of vertical lines represent a transient (it is shown here for illustrative purposes), animated trail showing that the BUG link refers to the BUG object.

If an object is already shown on the screen, and the user chooses Show again (whether from the original link that brought up the object or from another link of the object, say, embedded in some object), then the animated trail will trace the path from the link to the object shown. That is, the user cannot show one object at more than one place on the screen, following the direct manipulation paradigm.

Three fields are filled by the machine when an object is first created:

- Date: field: This is filled with the date and time of the creation.

- Creator: field: This is filled with the name of the user (or in the future, the agent) who has created the object. Objects that are not created by the user can be displayed like other objects; however, they cannot be changed.

- Name: field. The names of objects are used to test whether they are the same object (or different versions of the same object). Therefore, the system needs a unique name for each object. If the user does not give a name, Object Lens generates a "useful" name for the object. The algorithm for name generation is as follows:

  - Object Lens tries to generate a useful name: For example, nameless message objects get the To: field and its value(s) as their names. Description objects use the following algorithm:
    - if there' is a user provided name, use it.
    - otherwise, get a concatenation of fields and values.
  - if there are no filled fields, use the form "Any <type of description>" (*e.g.* "Any Bug" or "Any Message").

  If the machine generated name is too long, it is truncated to a reasonable length. If the generated name is an empty string, then the string "Nameless" is used.

  - If the machine procedure cannot arrive at a unique name (*e.g.* the name it generates is already taken), then a global counter is concatenated to the machine generated name. For example, the first nameless MESSAGE to "Kum-Yew Lai" gets the name "To: Kum-Yew Lai." The second nameless

MESSAGE to "Kum-Yew Lai" would make the machine generate the same "To: Kum-Yew Lai 1" instead.

Object Lens helps the user to construct an object by providing, for each field, a default value, an explanation of what the field means, a set of alternatives, and a set of descriptions; please see Figure ?. Again, following the general interpretation of the mouse buttons with respect to selection, the left mouse button replaces the current value(s) of the field with the selected value; the middle mouse button adds the selected value to the current value(s). This is true for the Default, Alternatives, and Descriptions options.

We have already described what the Default option does.

The Explanation option provides a brief note on the meaning of the field. This explanation, as with other prompts (eg. asking the user to place a region), is printed in the prompt window (the topmost window) of the object. In general, the prompt window for an object is where the system will print messages about actions called from the object; similarly, it is where the user can type her inputs if needed. In cases where there are no prompt windows (eg. messages for actions called from links), the InterLisp prompt window is used.

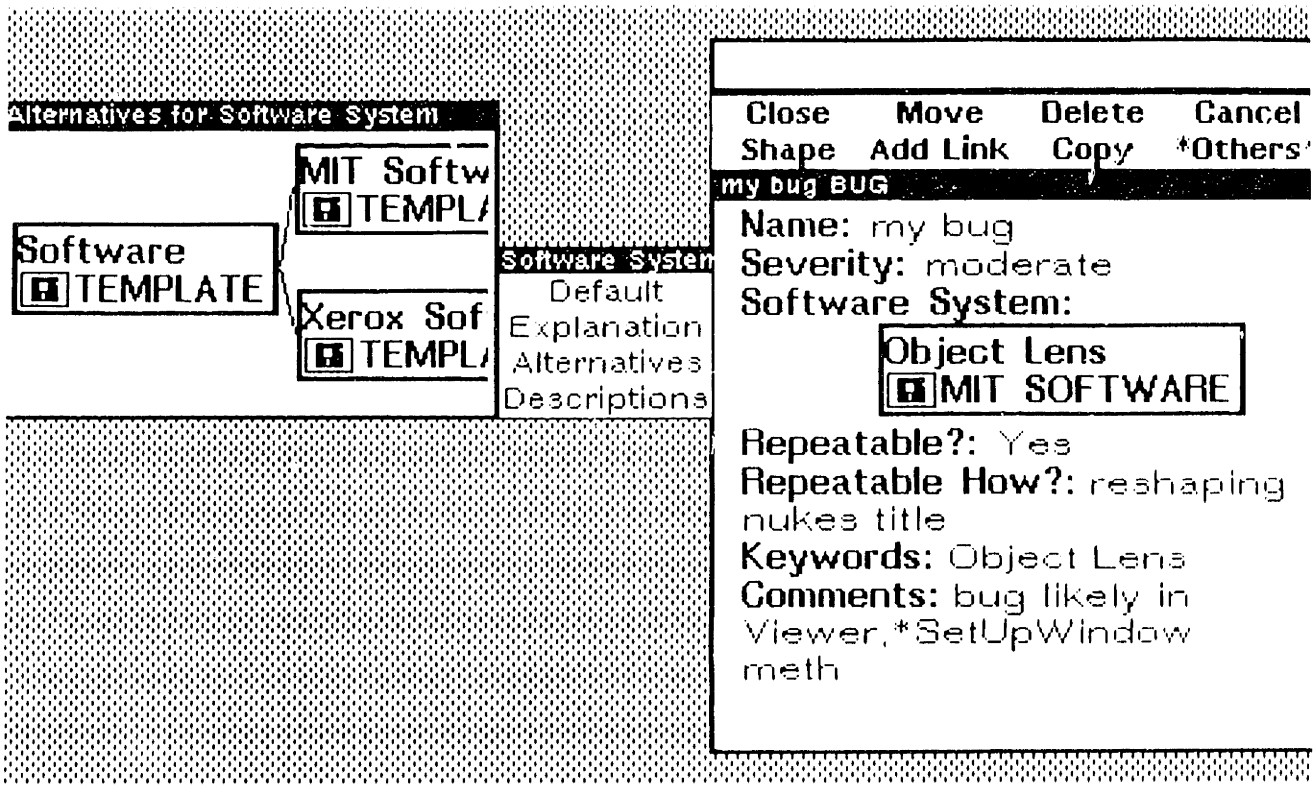


Figure 8: The Alternatives for the To: field. The user adds an MIT SOFTWARE object to the Software System: field, and call it Object Lens.

The Alternatives option brings up a set of alternatives that may be entered into the field. The Alternatives may be a list of strings, in which case a menu is popped up. The Alternatives may also be a template (eg. the SOFTWARE template). In this case, a tree is popped up. This tree consists of the root template and its subtemplates. Clicking left on a template in the Alternatives graph creates a new object of that template type, and replaces the current value of the field with a link to the new object. Clicking middle adds a link of a new object to the current value(s) of the field. If there are no Alternatives listed, but the Descriptions aspect has a value (which must be a template), then this value is used just as if it is in the Alternatives aspect – *i.e.* an Alternatives graph is produced, consisting of the template in the Descriptions aspect and the subtemplates.

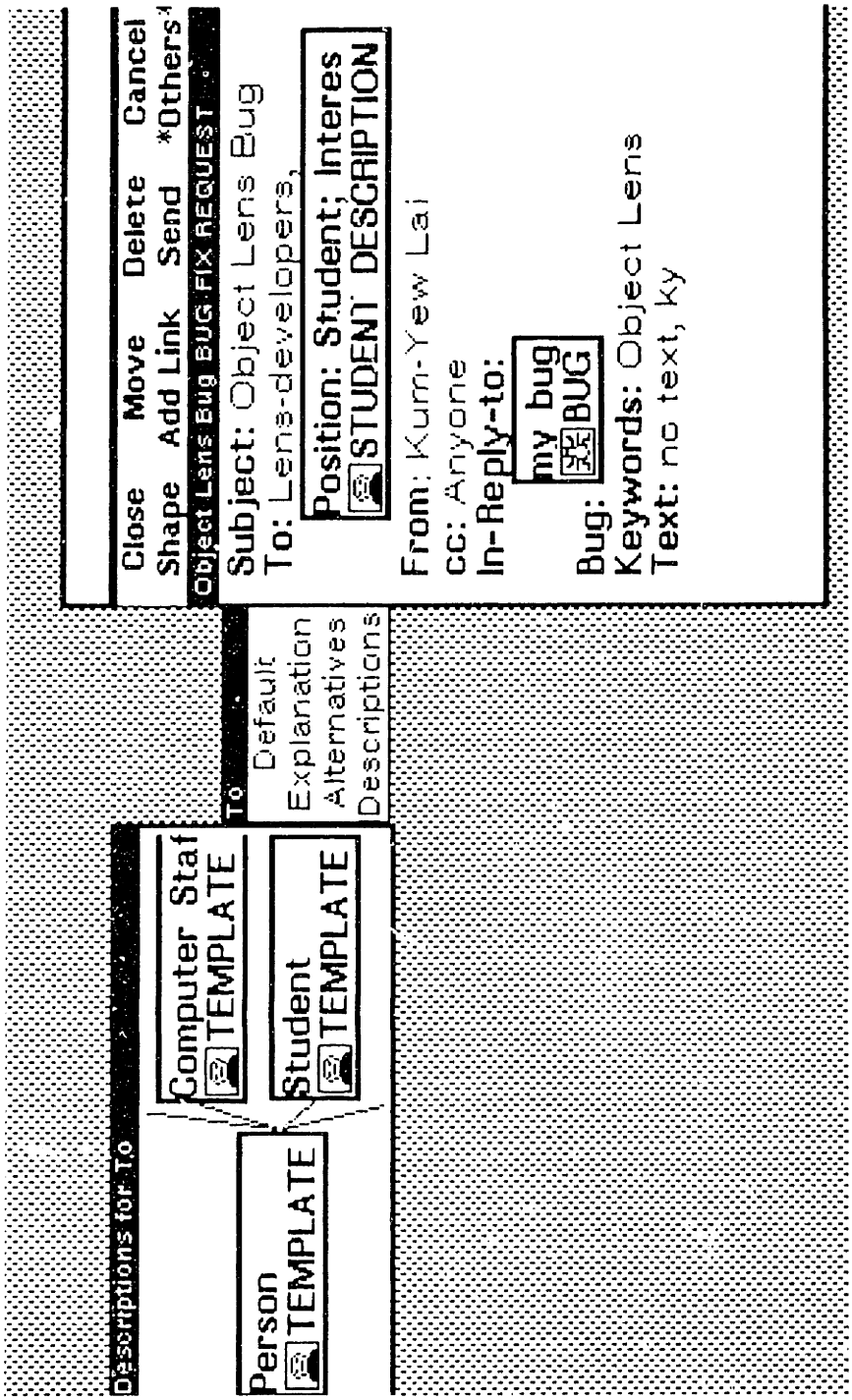


Figure 9: The Descriptions for the To: field. The user adds a description of STUDENT to the default vaue of Lens-developers.

In some cases, the user may want to enter a virtual subclass of objects, rather than specific objects. The Descriptions option is like the Alternatives option in that there is a root template, and a similar tree is shown. However, the new objects created are of a special template type whose name is a concatenation of the name of the selected template and the string "DESCRIPTION." For example, if the

user mouses left on a STUDENT template, an object of template type STUDENT DESCRIPTION is created, and its link replaces the current value of the field. An object of the STUDENT DESCRIPTION template has the same fields as the STUDENT template. A description represents a virtual subclass of objects which satisfy the values of its fields. Figure ? shows a case where the user adds a description of students to the default value of Lens-developers in the To: field. The description has its name generated by Object Lens; this description object is shown in Figure 10. An empty field means that the value is implicitly the most general object that can be filled in that field. If the user wants to indicate that no value is to be in that field (*e.g.* the Bug: field os a message must not have any value – that is, there is no bug), then she should put in the system object called "None".

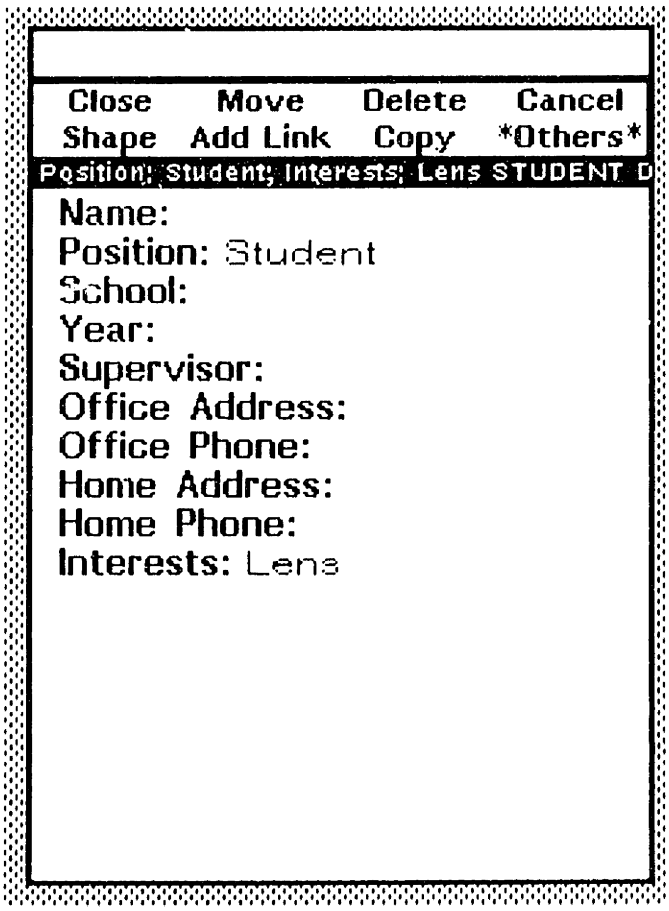


Figure 10: The "Position: Student; Interests: Lens" STUDENT DESCRIPTION object. The School: and Year: fields are local to the STUDENT template. The other fields are inherited from ancestral templates.

A subsequent section on the use of descriptions desribes how this matching of an object against a description is done. The value of the Descriptions aspect

indicates the most general description of the objects that can be the values of the field. Therefore, it serves the same function as the value restriction in knowledge representation languages like KL-ONE (Brachman and Schmolze, 1985). However, unlike these languages, the value checking in Object Lens is more flexible; this is discussed in a subsequent section on value checking.

THE PRESENTATION OF OBJECTS

Earlier, we discussed the several aspects of an object. The Customize Name, Show Link Object, Show in Forms, Show in Tables, and Show in Networks aspects are collectively called Display Format aspects. They control how objects may be displayed. To change any of these aspects, the user chooses Change Display Format in the action menu on the top of a displayed object. This pops up a menu that allows the user to choose specifically which Display Format aspect she wants to edit. After that, an aspect editor is displayed in the region specified in the Aspect Editor Region: field of the Customizer object. Choosing Close saves the aspect values and updates the display of the corresponding object using the new aspect values.

Both objects and templates have all the aspects described earlier. All the aspects can be changed only in the templates. Objects get their aspect values through inheritance from the templates. However, the user can also change the Show in Forms aspect at the object level, as opposed to the template level. This means that the user can display (or do not display) some fields for a particular object, while not letting this customized display propagate to other objects.

As mentioned earlier, the Descriptions aspect for a field indicates the kind of objects that can be put into the field. For a FOLDER object, this is equivalent to a restriction on the kind of objects that can be in the Contents: of the folder. The display format for the values of the Contents: field of a folder is controlled by a virtual object of the Descriptions aspect template type. For example, the value of the Descriptions aspect of its Contents: field of the New Mail folder is the MESSAGE template. Therefore, the virtual object is a MESSAGE object. The aspects that are relevant for virtual obejcts are Show in Tables and Show in Networks. Initially, the aspect values of the virtual object are inherited from the MESSAGE TEMPLATE. However, the user can change the values of the aspects of a virtual object to customize th display format for a particular folder. This change will not influence

the values of the aspects of other MESSAGE objects. Therefore, the display format of other folders with MESSAGE virtual objects are unchanged. This is unlike changing the aspect values at the MESSAGE template, where the aspect values are propagated to all MESSAGE objects, virtual or not.

Each Display Format aspect is described below:

- Customized Name. Each field may be renamed by the user. Choosing the Customized Name aspect after choosing the Change Display Format action on a template puts an aspect editor on the screen. Figure ? shows the aspect editor for the MESSAGE template. Notice that the Name: field has been renamed to Subject: and the Comments: field to Text:.

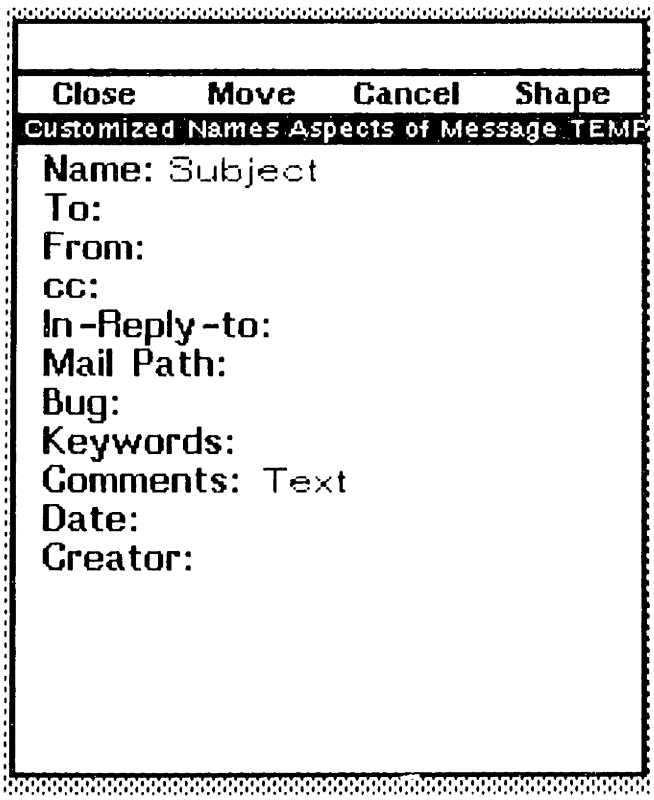


Figure 11: The Customized Names aspects of the MESSAGE template.

- Show Link Object. When a link to an object is a value for a field in another object, we say that the first object is embedded within the second. When an object with embedded objects is displayed on the screen, the embedded objects may also be automatically displayed. This automatic display of embedded objects is done if the value of the corresponding field of the embedding object has a value of "Yes" in its

Show Link Object aspect. For example, in Figure ?, a RULE object has an embedded MESSAGE DESCRIPTION object in its If: field and an embedded ACTION object in its Then: field. In Figure ?, the If: and then: fields of the RULE template have "Yes"'s for their Show Link Object aspects, but the other fields have "No"s. Displaying the RULE object will also automatically display the embedded MESSAGE DESCRIPTION and ACTION objects, but not the links (if any) in other fields.

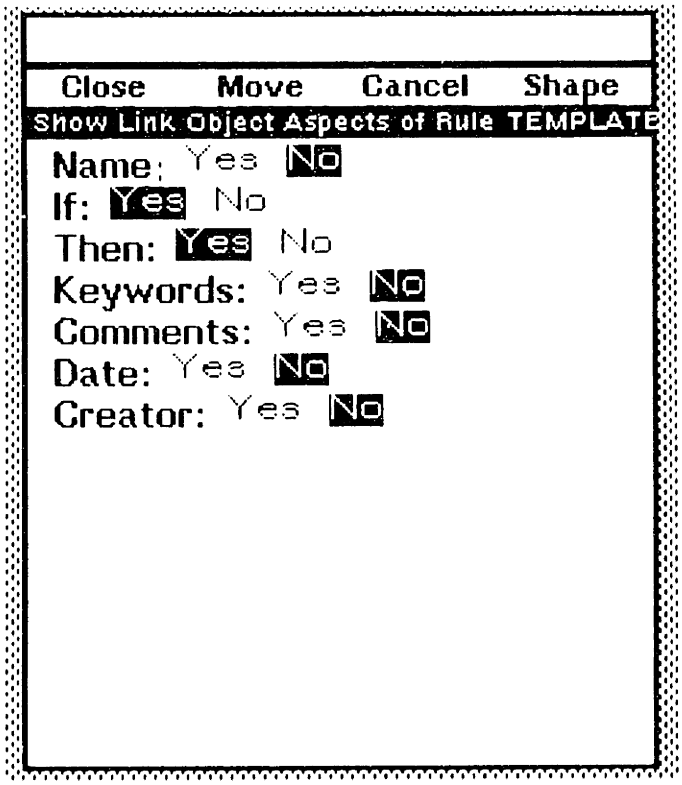


Figure 12: The Show Link Object aspect of the RULE template. Links in the If: and Then: fields automatically show their objects when they first get into the either field (*e.g.* the user adds an embedded object into either field), or when a RULE object with embedded objects in either field is displayed.

The same criterion for automatically display embedded objects applies when an object is first embedded. This happens when a user uses the side menu to put in new links, when the user copy selects from any existing link or object (discussed later), or when the user uses the Add Link or Grab Link actions (also discussed later). In the RULE example above, if the user replaces the MESSAGE DESCRIPTION with another description, Object Lens will automatically display the new description. However, replacing the embedded ACTION object with a new ACTION object will not cause the new ACTION object to be expanded.

• Show in Forms. Each field has a value of "Yes" or "No" for this aspect. Figure ? shows the aspect editor for the MESSAGE template. It shows that Date: Mail Path:, and Creator: fields are not shown. As mentioned earlier, the user can change this aspect at the object level. Therefore, a user can specifically change the aspect values for a MESSAGE object so that she can see the value of the Mail Path: field for that particular object.

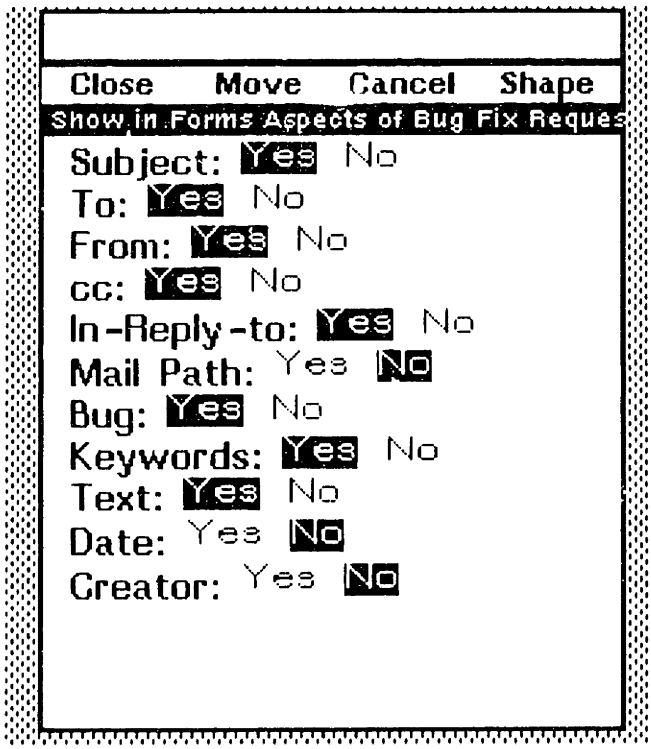


Figure 13: The Show in Forms aspect of the BUG FIX REQUEST template. The Mail Path:, Date:, and Creator: fields are not shown. The user can also change this aspect at the object level (*i.e.* change the aspect for a particular BUG FIX REQUEST, not to the template).

• Show in Tables. This aspect (and the Show in Networks aspect) is used only for virtual objects controlling the display format of the contents of folders. Figure ? shows a FOLDER object called My meetings. Currently, the links in the folder are displayed in the order in which the link enter the folder. Of course, the user can rearrange the order. This is done using a "pull" operation similar to the moving of text within a form. The user first selects the rule after which the moved rule should go. She then holds down both the CTRL and SHIFT keys and clicks the left mouse button on the rule to move. The column widths of a table can also be changed. As before, the user holds down both teh CTRL and SHIFT keys. This

time, she uses the middle mouse button to drag the space between two columns to the left or the right.

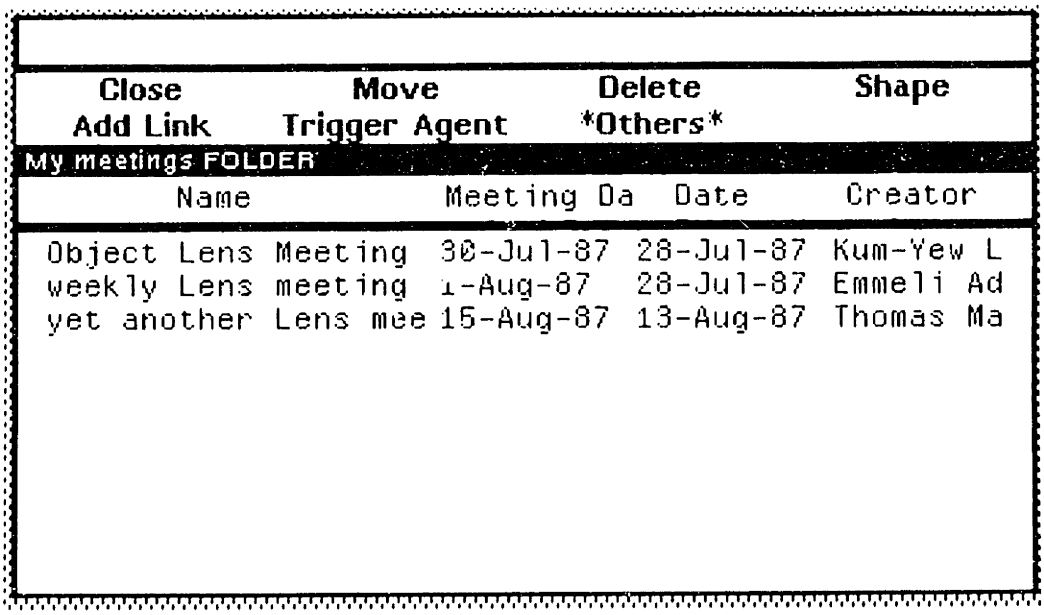


Figure 14: The My meetings FOLDER object.

The Show in Tables aspect of the My meetings FOLDER object is shown in Figure ?. The Descriptions aspect of the Contents: field of the folder is a MEETING template – only links to meeting objects can be values of the folder contents. The aspects editor shows the fields of a virtual object of the MEETING template. We cannot change the Show in Tables aspect of the MEETING template directly, since that would propagate the changes to all folders with MEETING templates in their Descriptons aspects. For the sake of simplicity, the title of the aspect editor is "Show in Tables Aspects of My meeting FOLDER."

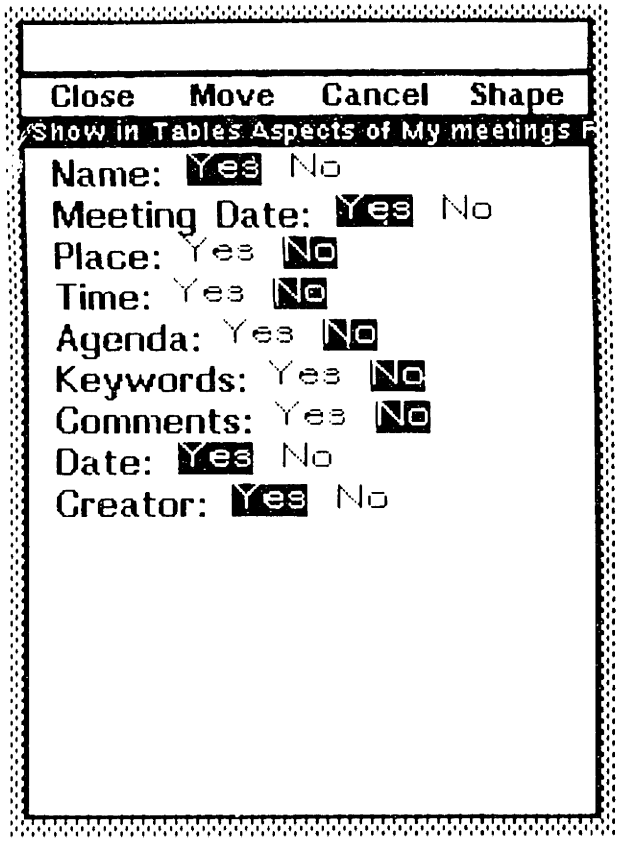


Figure 15: The Show in Tables aspects of the My meetings FOLDER object.

• Show in Networks. This aspect, like the above, changes the display format of a folder. If a field has a value of "Yes" for this aspect, then it is used as a pointer from one node of the network to another node, For example, the user can display a conversational network of the links in the New Mail folder by displaying the Reply-to: field. The network will show all the links and how they are connected to each other using the In-Reply-to: field; Figure 16.

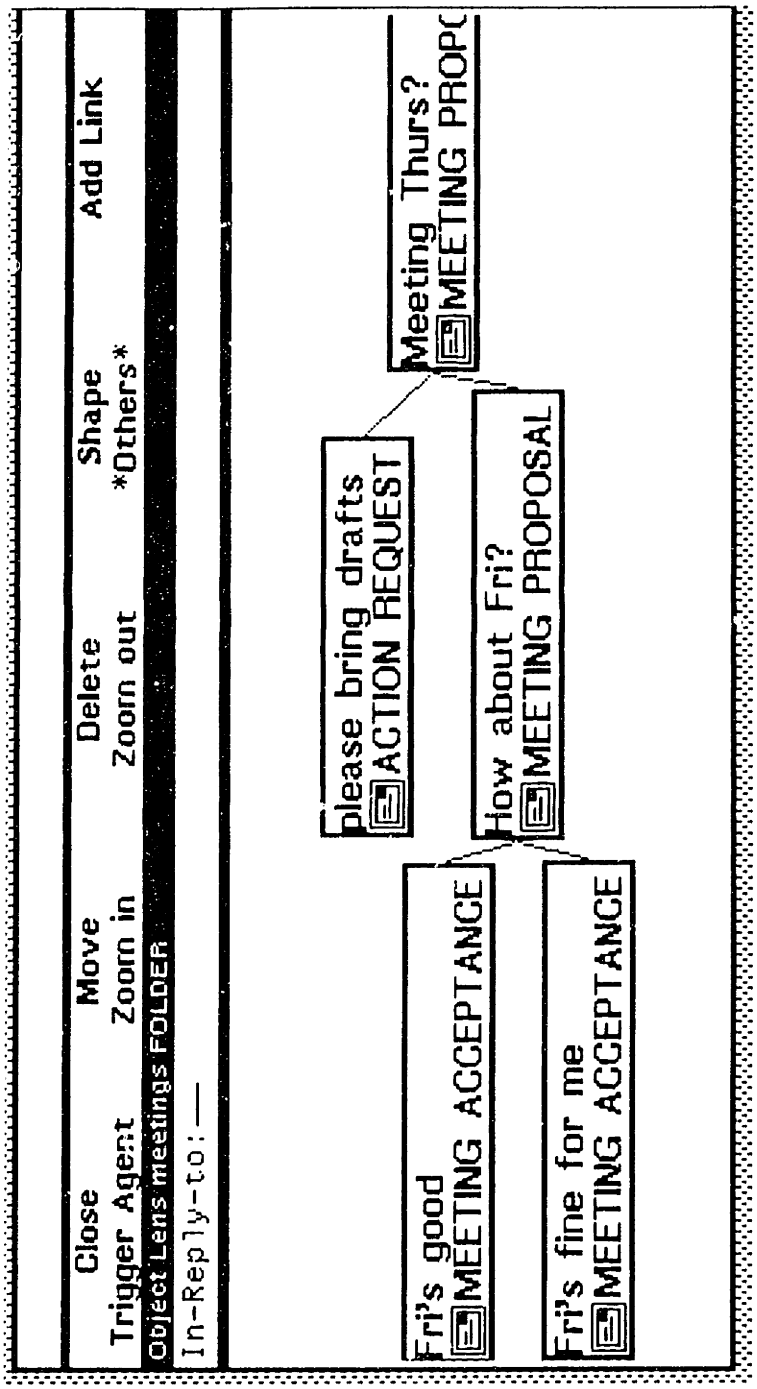


Figure 16: The Object Lens meetings FOLDER. This folder shows the In-Reply-to: fields connecting the links in the folder.

If a FOLDER object has its contents shown in a table, choosing Show in Networks shows the contents as a network as well as bring up the aspect editor. Conversely, if the contents were shown in a network, choosing Show in Tables changes the display to a table.

We have discussed how objects can appear on the screen. We now describe how objects are placed on the screen.

Let us consider the case where the value of the Overlapping Windows: field is "Yes." This means that if there is a stored region for the template type of an object, then all objects of that template type are placed in that region. For example, if the stored region for the RULE template is in a pre-specified region, then all RULE objects are placed in stacked up in that region. If there is no stored region, then the user is asked to shape one. This is stored, and subsequent objects of the same template type are stacked in that region. Reshaping or moving any object stores that region as the new region. Objects will then be stacked into this new region.

Now, we consider the case where the value of the Overlapping Windows: field is "No." Suppose the user wants to display a RULE object. If there is a stored region, and no RULE object is already displayed, then the stored region is used to display the current RULE object. If there is already a RULE object displayed, then it must have been displayed using the stored region. The current RULE object will not be displayed on top of this displayed RULE, as in the previous case of overlapping windows. Instead, the user is provided with a stretchable ghost box which is of the same size as the stored region, and is asked to place the current RULE object using this ghost box. Now, this new region will be the stored region. As before, reshaping or moving any object stores that region as the new region.

In both cases above, the regions are stored when an object is displayed, not when it is closed. Also, the two cases apply only when the Show action is not called from an embedded link. Otherwise, the object is displayed to the right of the containing object. Of course, this applies only if the embedded object is not already displayed (otherwise, as mentioned earlier, Object lens will just show the animated trail from the embedded link to the object already displayed). If the displayed object goes out of the side of the screen, it will be pushed back just enough to remain on the screen. The region of the displayed object is not saved, since it is only relative to the containing object.

Finally, if the containing object is closed, the displayed embedded object will be closed automatically if it has been automatically shown earlier (*i.e.* because the Show Link Object aspect for the appropriate field is "Yes").

ACTIONS ON OBJECTS AND LINKS

The actions for an object are grouped into three categories:

- those applicable to how the object is shown on the screen (eg. Cancel, Close, Delete, Move). These, together with those actions in the next category, are shown at the top of the object when it is shown on the screen.

- other actions applicable to the object (eg. Add Link, Change Display Format, Change Template, Copy, Hardcopy, Show History, Shrink). Some of these actions are less frequently used (eg. Change Template, Hardcopy, Show History) and are shown in a pop-up menu when the user chooses *Others*.

- those applicable to the links of the object (eg. Copy, Delete, Move, Show Object). These are shown in a pop-up menu that appears after the user points the mouse at the link and clicks the right mouse button. Clicking the left mouse button directly chooses the first action listed in the pop-up menu; and clicking the middle button chooses the second action listed. For now, we have Move as the first action, and Show as the second.

The above categories are not hard and fast, since one can also put actions applicable to the object in the pop-up menu obtained by mousing on a link to that object. For example, Hardcopy could appear in the pop-up menu; but it is put there because the user usually wants to see the object before deciding whether to make a hardcopy of it – she may change the display format, for instance. The primary purpose of the categories is to avoid showing too many actions to the user at once. When it is necessary, we do not insist on keeping the categories. For exmaple, Create Object is available in the pop-up menu when the user mouses on the link of a template. This is so, despite Create Object being an action applicable to the template (not the link to the template), because it makes it easier for the user to create new objects, without going through the unnecessary step of showing the template itself.

The maximum number of actions, including *Others*, shown is ten (this is usually two rows of items). Others have to go under the *Others* item.

There are some actions that do different things depending whether the action is executed on a link or an object. For example, Move and Delete are different operations:

- Move
  - called from the action menu of an object: moves the object.
  - called from the pop-up menu of a link: moves the link.
- Delete
  - called from the action menu of an object: deletes the object, after a confirmation from the user. The screen is also updated to reflect the deletion of this object.
  - called from the pop-up menu of a link: deletes the link. If this is the last link to the object, and the object is referred to by another object, then a link will be placed in the Orphans folder. Incidentally, Sending a message object makes a copy and sends it away. It also closes the local object. Therefore, if there are no links to the local object, a link to it will be placed in the Orphans folder.

The following is a brief description of some of the more interesting actions available on objects and links. A complete description is in the appendix.

The actions are "push" operations if the destination is the screen, and a "pull" operation if the destination is another object. For example, the Move action on a link (or an object) is a push, since the user first selects the link (or the object) and then the new position. On the other hand, the Add Link action on an object is a pull, since one first selects the destination (a position within the object), then the action, and finally, the links to go to the destination. The goal for these "push" and "pull" schemes is to combine the advantages of both schemes without causing confusing inconsistencies. The "push" operations are used for putting objects or links on the screen (*e.g.* in Moves and Shapes). This gives a direct manipulation and physical interface to the objects. The "pull" operations are used for getting links into the object in the current focus of the user (*i.e.* the object the user is

editing). Users normally follow a chain of links when they look at objects – displaying an object, then displaying the object pointed by an embedded link in the first object, and so on. Therefore, the "pull" operations are a natural way to construct these chains. The "pull" scheme is also consistent with the "pull" operations of the side menus (the user selects the field – the destination – and choose what to get into it). Besides these conceptual arguments, the "puli" scheme has other advantages. From an implementation point of view, it is easier to build "pull" operations given that TEdit, the primary text editor in the Xerox 1100 series machines, supports these kinds of operations. Many users have grown familiar with the "pull" operations in the InterLisp world because many tools like NoteCards (Trigg, Suchman, and Halasz, 1986) and Lafite or GrapeVine (the primary mail systems used on the Xerox machines).

The most reasonable alternative to having both "push" and "pull" operations is to support just "push" operations. For instance, the Add Link "pull" operation would be changed to a Copy, so that a user can mouse on a link and copy it into an object. Similarly, the Grab Link "pull" would be changed to a Move, so that the user can move a link into an object (as opposed to just onto another part of the screen). This alternative has the appeal of requiring fewer commands – just the Copy and Move would do most of the work. However, it becomes unnatural to build hypertext. It is also cumbersome if the user wants to add an object into a folder.

Just as there are some actions that are rarely used and therefore hidden under the *Others* menu item, there are some combinations of actions that are frequently used, and are provided as menu items. For example, looking at the rule folder of an agent object by calling from the link of the agent (please see the subsequent section on agents for more details) is a frequent operation, so we provide a Show Rule Folder action for agents, and make it available in the pop-up action menus of agent objects. Another example is the Add Object action for folders. This creates a new object and puts a link to it into the folder. The template type of the object is either the template of the Descriptions aspect of the Contents: field of the folder, or a subtemplate of that. This is really a special case of the usual addition of a link to a field – if the Contents: field were explicitly shown as in a form, then this action corresponds to buttoning the field, getting the Alternatives graph, and choosing to add a object of the selected template type in the graph. Since the Contents: field is not shown explicitly, the user would have to explicitly create a new object from the User Templates folder (or make a copy of an existing

object) and use the Add Link action to get a link into the folder. The Add Object action is a macro that does this easily.

INHERITANCE

The general scheme for inheritance states what is to be inherited, along what edge between two objects (an edge between two objects is another way of saying one object is in the value of the field of the other). The current implementation has only built in inheritances. These are:

- inheritance of all field values and their aspects along the Membership: (or instantiation) edge. For example, an object inherits all field values and their aspects from its template object. The Membership: edge is one between a template and an object.

- inheritance of all field values and their aspects along the Subtemplate: (or subclass) edge. For example, the Message template inherits the Response Types: field from the Thing template. The Subtemplate: edge is one between two templates.

- inheritance of the Fields to Show: field along Subfolder: edges. Therefore, Folder objects inherit their display formats. The Subfolder: edge is one between two objects.

In the future, we may implement a Subagent: field that allow agents to pass on values of their Automatic Triggers: and Rule Folder: fields. One complication with this is that it is not clear in which order the Rule objects should then be fired.

CHANGING OBJECTS AND TEMPLATES

Users can also change the template of an object, *i.e.* change the fields and inherited values and aspects. This is easily accomplished using the Change Template action. Object Lens will try to retain as many fields as possible, based on the field names. Current fields which do not exist in the new template are put into the Comments: field. This means that the template of the object could be either "higher up" in the template hierarchy, or "lower down."

The templates themselves can be changed by adding and deleting fields, using the Add Field and Delete Field actions. Such changes are propagated to existing objects of the template, including the descriptions (eg. those in the rules).

SEMI-FORMAL CHECKS

If a value of a field is expected to be an object, then if the value is a string, that string will be treated as the object with it as its name. Whether a value is expected to be an object or not depends on the value checking described below.

The value of the Descriptions aspect of a field must be a template. This template serves as a value restriction on the values of the field. The checking of the value restriction span a spectrum of tightness:

- Equality: All values of a field must be strings, and these must be members of the list of Alternatives. For example, the values of the Overlapping Windows: field of the Customizer object must be one of "Yes" or "No." If there are only a few alternatives, then they are displayed as buttons for the user to select.

- Membership: All values of a field must be strings or objects, and these must be either members of the Alternatives, or be subsumed by the Descriptions.

- No checking.

The Alternatives, if they are a list of strings, can be thought of as virtual objects: they are not instantiated from a template, but they can be equated with other objects/strings (using their names) and they can be checked using subsumption. One can think of them as intermediates between the unstructured strings and its formalization as an object. Similarly, descriptions are virtual templates: they denote the set of objects which their descriptions subsume. It is natural to think of descriptions as entities which are less formalized than shared templates used by groups of people. The Change Template action allows users to change templates not only "horizontally" (ie. change an object from one template to another), but also "vertically" (from an object to a description, or to a template).

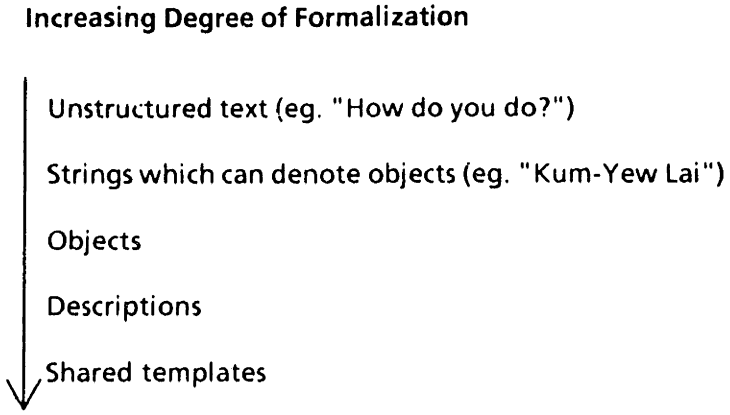


Figure 17: The progressive formalization from unstructured text, to strings that could be interpreted as objects, to descriptions (which are not formally shared between users), to formally shared templates.

DESCRIPTIONS

Descriptions are useful in capturing abstract sets of objects. They are most used in rules, described below.

An object is said to match a description if both of them are created from the same template type (*e.g.* a MESSAGE object and a MESSAGE DESCRIPTION description are both created from the MESSAGE template) and if the values of the object satisfy the values of the description for each field. The values in a description can be combined using boolean combinations, and using either infix or prefix notation. Besides the usual "OR", "AND", and "NOT", commas are treated as disjunctions, ampersands as conjunctions, and tildes as negations. The fields taking dates can also take description values like "> TODAY" (TODAY, YESTERDAY, and TOMORROW are system words dynamically bound to the appropriate dates).

The value of the Name: field in a description is used in matching. It is not used as an identifier of the description. The identifier is machine generated, and consists of a concatenation of the field names and their values in the description. As in other objects, if the machine procedure for generating names does not arrive at a unique name (*e.g.* there are no values filled for the descripton), then the date is used.

Descriptions have an action called Resolve. This prints the links of those objects that satisfy the description into the Lens prompt window. Figure 18 shows

the resolution of the STUDENT DESCRIPTION shown earlier in Figure ?. The Prompt Window object can be scrolled forward and backward. Clear clears the history of printouts to the window. The Prompt Window is used to output the results of actions to objects (*e.g.* Show History, which prints out a list of RULE objects which have fired on a given object).

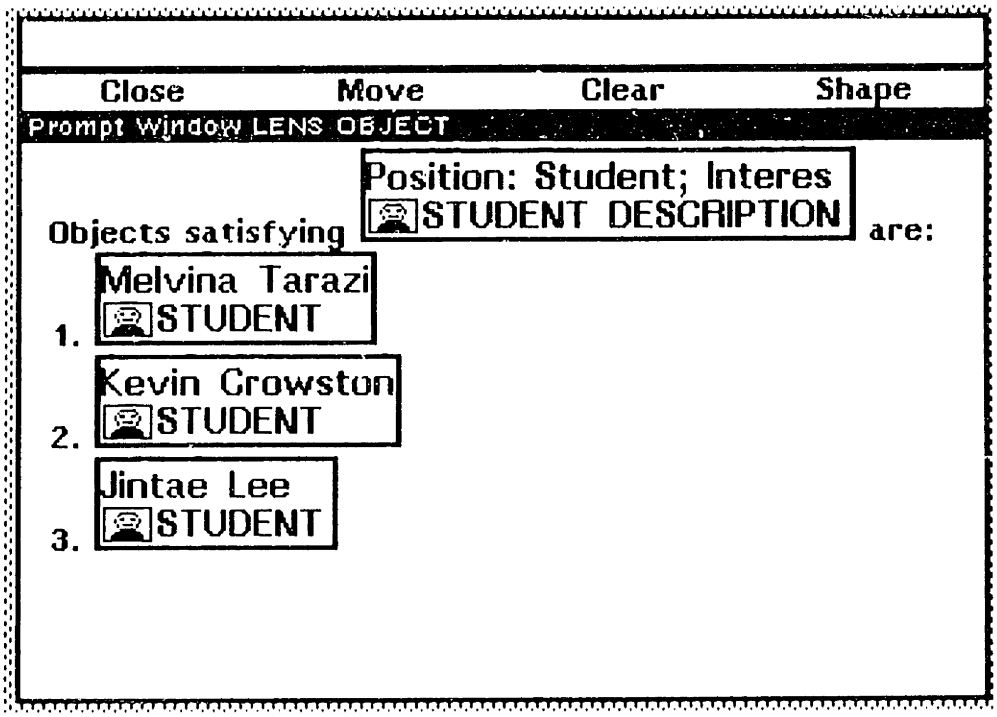


Figure 18: The Object Lens prompt window is a object of template type LENS OBJECT (which is not available to the user). The STUDENT DESCRIPTION has the value "Student" in the Position: field and "Lens" in the Interest: field.

AGENTS AND RULES

Agents are objects which do most of the work in Object Lens. Figure 19 shows an agent which runs its rules over the objects in the New Mail folder when new links to these objects enter that folder, or when the machine goes into Idle mode.

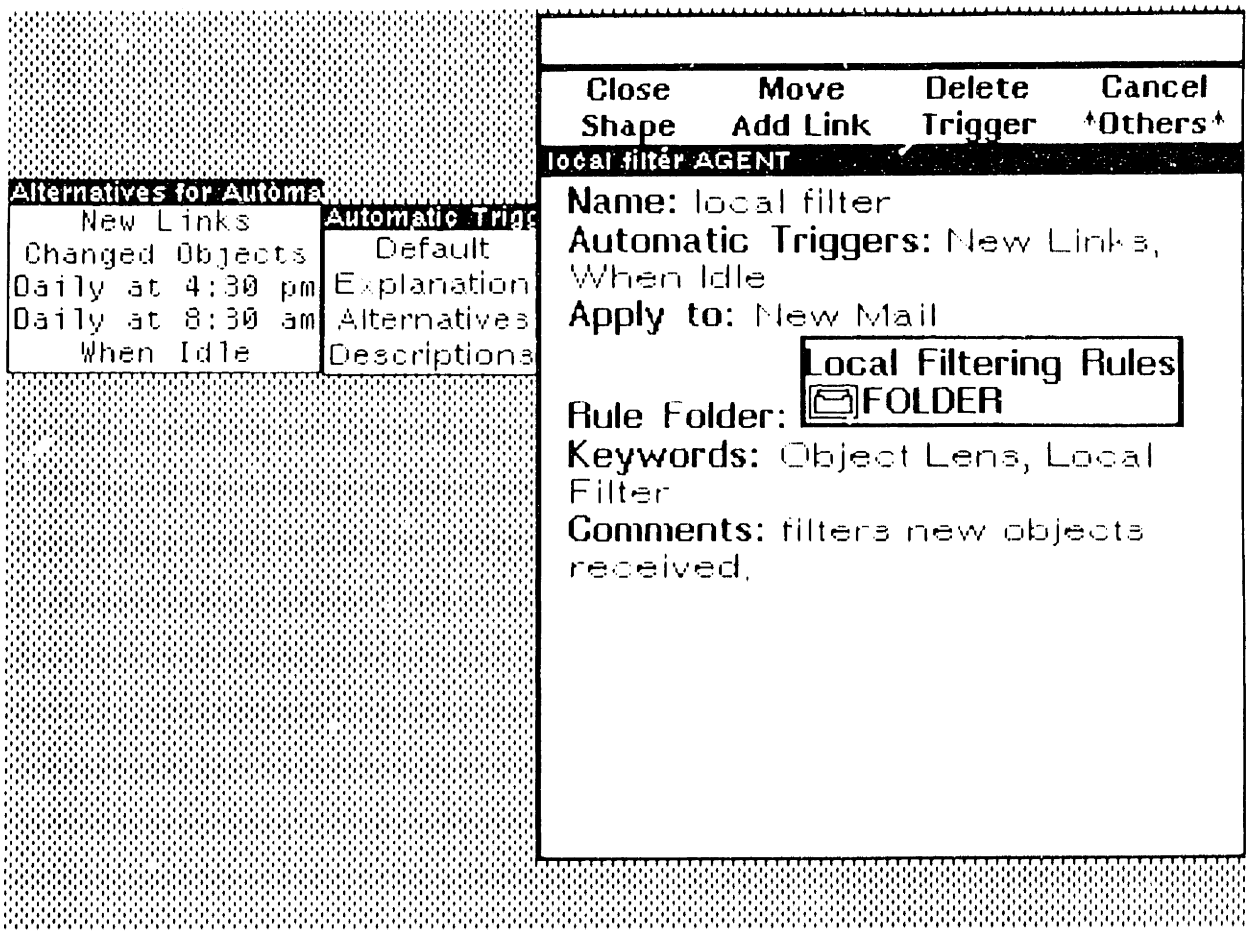


Figure 19: An agent which acts on objects in the New Mail folder.

An agent can also be triggered manually. One way is to call the Trigger action of the agent. Another is to trigger the agent from a folder. In the local filter agent above, we can choose the Trigger Agent action in the New Mail folder. This gives a pop-up menu of the names of agents which have New Mail as one of the values in their Apply To: fields. When an agent is triggered, it runs the rules in the folder in its Rule Folder: field. These are run as ordered in the folder.

The Local Filtering Rules folder is shown in Figure 20.

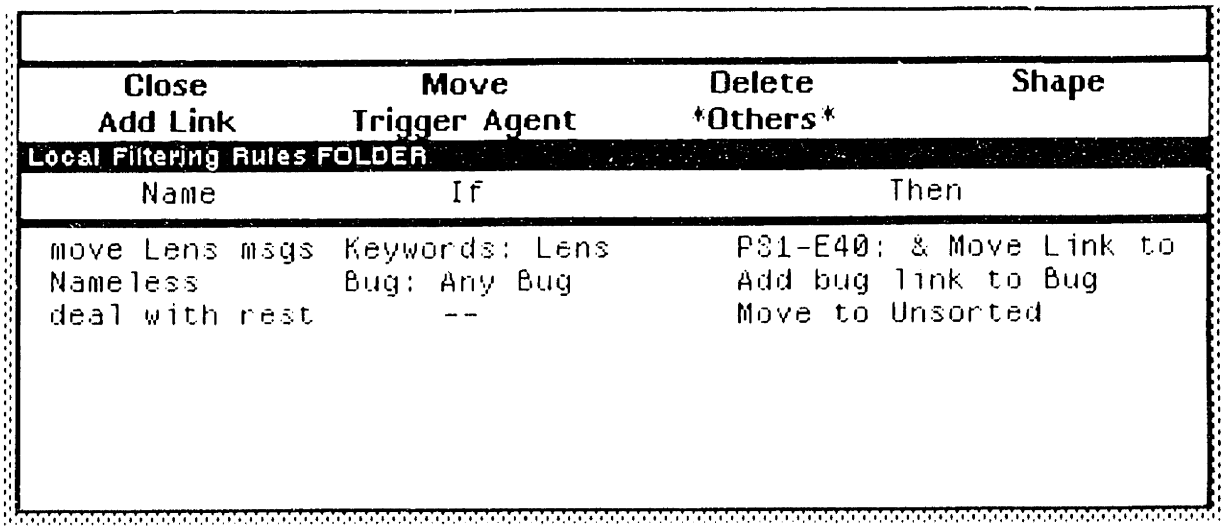


Figure 20: The Local Filtering Rules folder of the local filter Agent. Note that the second rule has been selected. Selected links are indicated by shading, just like those in folders displayed as networks.

We shall examine the three rules one after another.

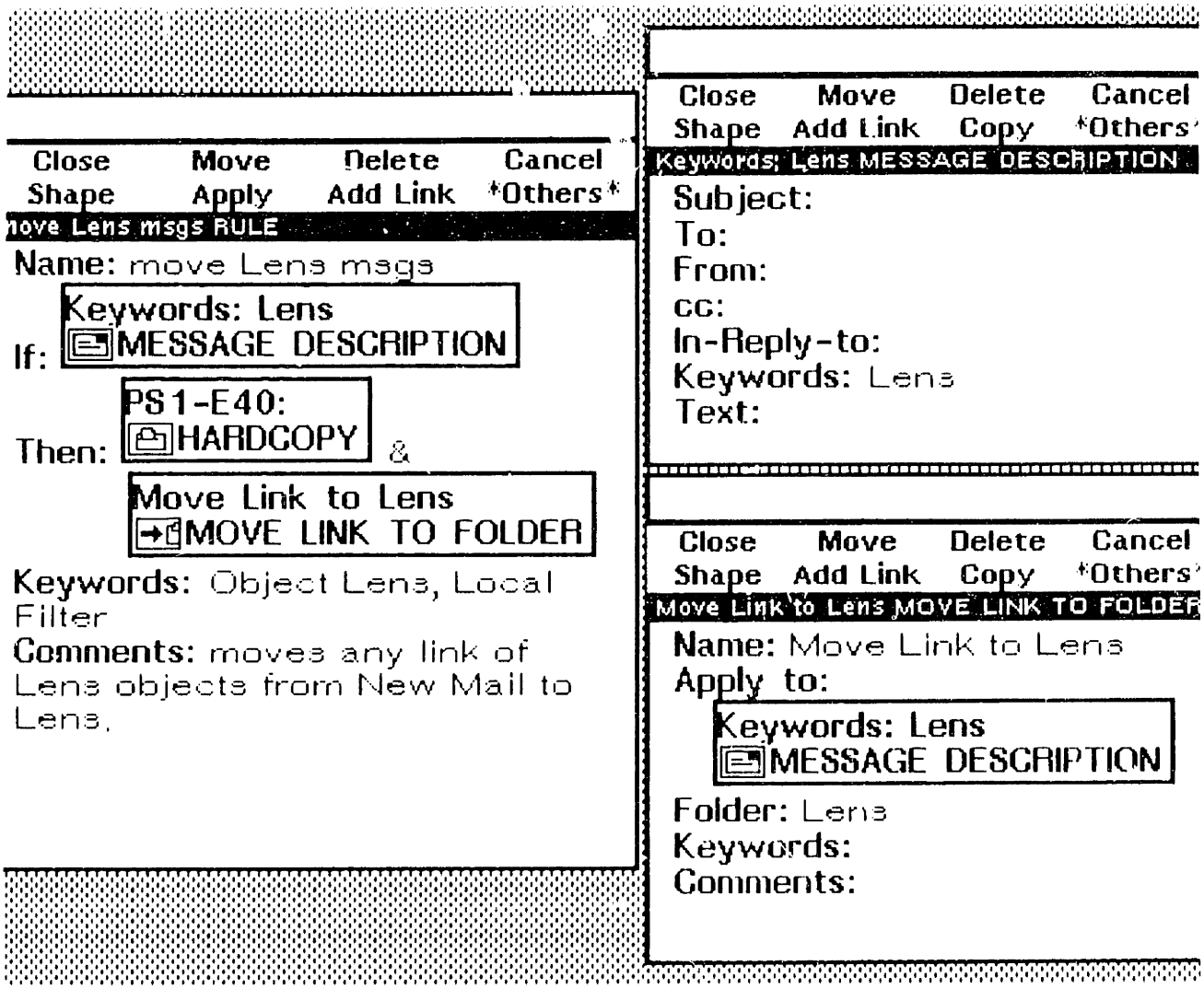


Figure 21: The move Lens msgs rule, the first one in the Local Filtering Rules folder. The actions Hardcopy and Move Link to Folder are conjoined.

The Apply action of rule objects is a pull operation. Like the Add Link ACTION for folders, the user selects the action, and then makes a selection on the objects on the screen using the mouse. The rule is then run over the objects in the selection. Indeed, the local filer AGENT executes the Apply action for each rule on each object in the New Mail FOLDER. The Apply to: field of the ACTION object defaults to the description in the If: field. Putting the description of the If: field in the Apply to: field of the ACTION object means that any object which satisifies the description will be used as the argument to the ACTION object.

There are two actions for the first rule: Hardcopy and Move Link to Folder. Both do not have user provided names, so Object Lens generates names for them. The Hardcopy action takes its name from the printer name as its name. The user has not changed this action, so the value of its Printer: field is the default printer. The Move Link to Folder action also has a machine generated name, which is of the form "Move Link to <folder in Folder: field>." This action object moves the link of whatever object that satisfies the description of the If: field to the Lens folder.

The second rule has a BUG description within a BUG FIX REQUEST description in the If: field. The ACTION object does not apply to the default bug desc in Bug Field description. Instead, it applies to the embedded description any bug. Any object that satisfies the predicate must have a BUG object that satisfies the embedded description. It is this BUG object that is used as the argument to the ACTION object. In general, the ACTION objects of a rule can apply to any arbitrary object in the workstation.

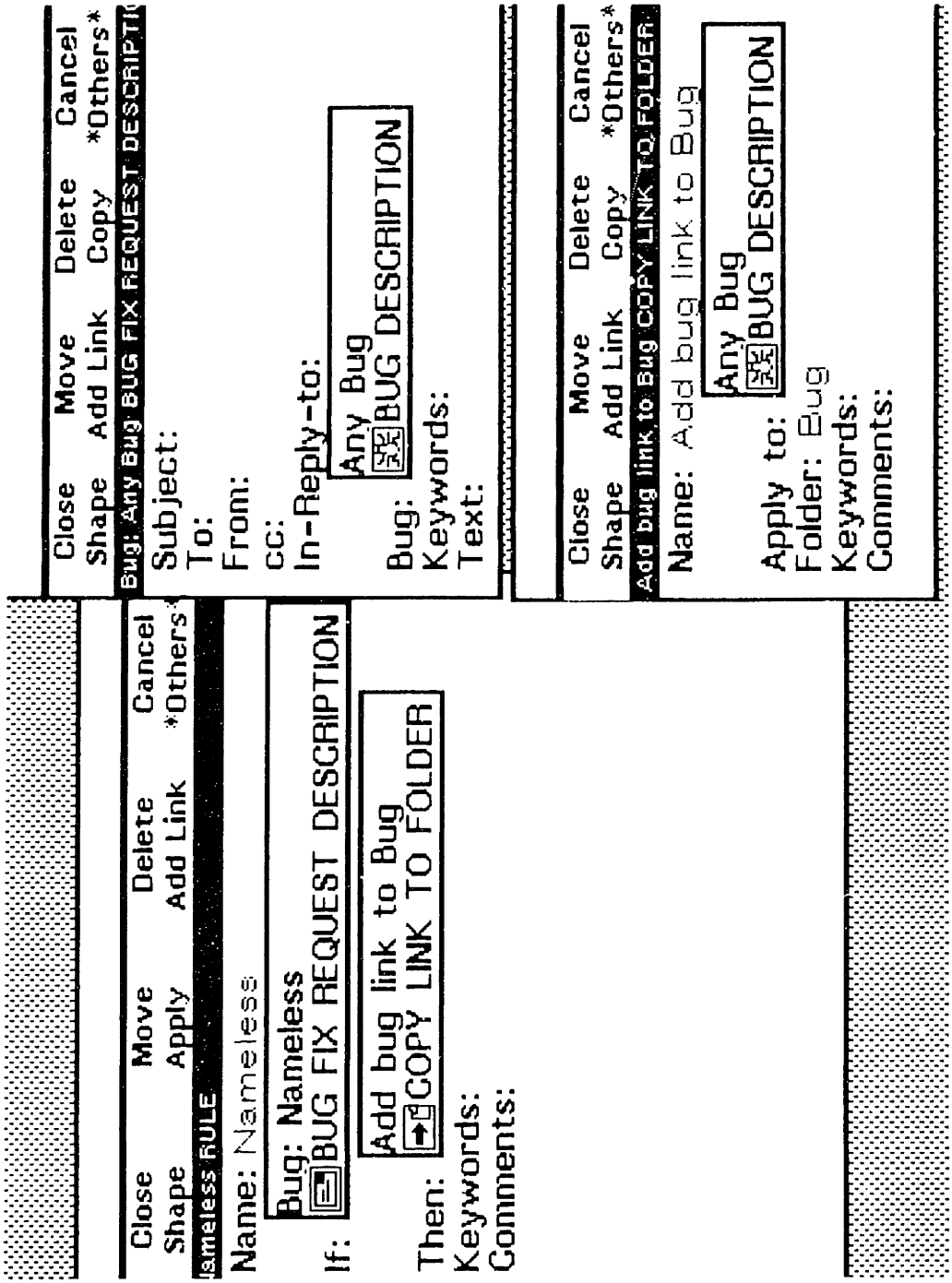


Figure 22: The second rule. Any object that satisfies the Any Bug description will have its link copied to the Bug folder.

Finally, the third rule has no value in its If: field. It applies to any object which still has links in the New Mail folder. Notice that unlike mail systems like Lafite, Object Lens deals with links in folders, not "real" objects. Moving a link in Object Lens deletes it in the current folder and puts it in the destination folder; in contrast, Lafite makes a copy of an object in the destination folder. Similarly,

deleting a link is really a delete, as opposed to putting a delete mark. Object Lens can afford to do this because it deals with links. Also, the user can recover from the most recent action by using the UNDO button. The side-effects of actions like deleting and moving links are explicit (*e.g.* not implicit like putting move and delete marks on the objects) and therefore close to a physical model (moving a link really moves it). Therefore, a rule like the third one applies only to those links that are still left in the New Mail folder after the previous rules have fired. Presumably, the user does not want to the second rule to delete the original link in the New Mail folder. Such cases may occur, say, when the user is not certain whether the rule completely specifies her criteria for moving BUG objects. Therefore, she leaves the original in the Unsorted FOLDER for future reference.

The actions applicable in a Rule object are exactly those applicable to the object in addition to three others discussed later. Therefore, one can put a Change Template action that translates one object type into another (one example of this use is that the user focuses on only a subset of the Templates provided by the system). The three extra actions are:

- the two "push" operations on links: Move Link to Folder and Copy Link to Folder. The latter two operations are provided especially for the processing of objects, since it is more natural to use "push" operations then (as opposed to their "pull" equivalents: Add Link and Grab Link).

- the Execute Lisp Code ACTION. This is not applicable to any specific object, but it allows the user to write customized actions on the object to be applied.

IMPROVEMENTS OVER THE LENS SYSTEM

Object Lens is considerably more complicated than the Lens system. It would be necessary to justify this increase in complexity. The improvements to the Lens system can be grouped into the following classes:

- Provide a unifying framework in representing messages and objects. This has advantages at different levels. At the implementation level, Object Lens has already shown that it can implement much more functionality with much less code. The amount of code in Object Lens is anywhere from a quarter to three-quarters of that of Lens, depending on the evaluation criteria. For example, Object Lens does

not use the Lafite mail system; the Object Lens code is only a fraction of the combination of Lens and Lafite. On the other hand, Object Lens has considerably more templates than Lens; including these will make Object Lens look bigger. In any case, the code for the core functionalities is about half of that of Lens. This reduction comes both from a cleaner organization (*e.g.* many object classes share a lot of code) as well as from more efficient coding.

Besides the implementation, a unifying framework also allows programming ease (there is less code to read, and there is a cleaner organization) as well as "conceptual hygiene" at the user interface level. An example of a cleaner user interface is that, formerly, the structure of BUG FIX REQUEST and BUG FIX ANNOUNCEMENTS objects share many fields that are really about BUG objects. With Object Lens, the structure of the BUG object is abstracted in the Bug: field of both the BUG FIX REQUEST and BUG FIX ANNOUNCEMENTS templates. Other examples of a cleaner interface are the aspect editors (which are consistent with the displayed objects – both use the form structure) and folders (which can be presented as networks or tables).

The user interface, however, is still rather unpolished. It would be cumbersome to use a clean but spartan interface. On the other hand, it is equally unsatisfactory to include too many *ad hoc* solutions (*e.g.* combining actions to crate action macros) to allow a more convenient interface. But this usually reveals the existing framework to be not clean, general, or simple enough. Therefore, the next prototype would be based on a cleaner, more general, and simpler framework. Hopefully, the development of the Lens system will follow such a trend of development; Figure 23.

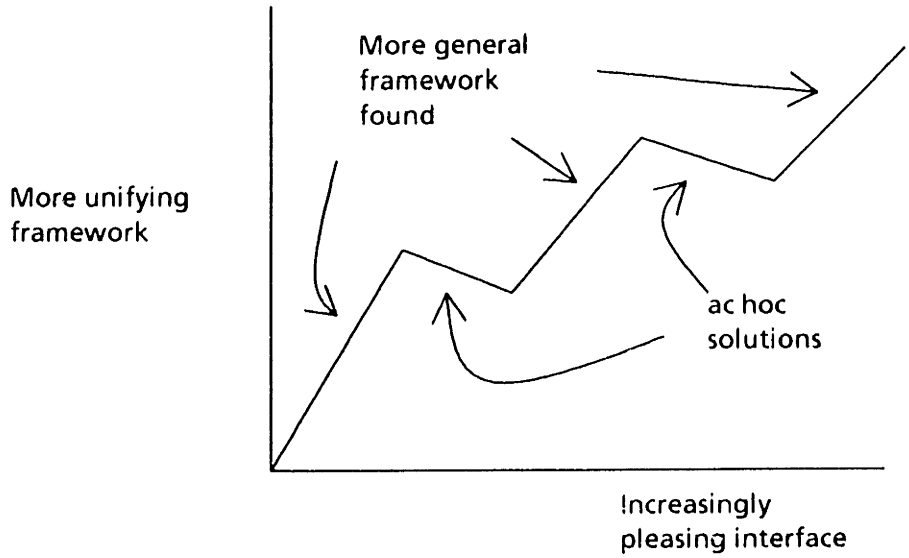


Figure 23: A possible development path for the Lens system.

• Reveal many of the *ad hoc* solutions to problems in Lens. Some examples of these are listed below:

  • in Lens, rule sets are attached to message types, so that the rules in a set apply to messages to the attached type as well as its subtypes. The conceptual problem with this approach is that is we view rules as yet another kind of objects, then confining rules to rule sets attached to message types is like confining our mail folders to be attached to message types. This means that NYT Article objects will go into a mail folder attached to the NYT Article message type; and so on for other types. Clearly, in the case of message objects, we want to allow the user to regroup her message objects in ways different that the types used for communication. Such new groupings in the folder network are orthogonal to the message type hierarchy, and are usually along dimensions like priority (*e.g.* an "Urgent" folder) or topic (*e.g.* "Washington News"). Similarly, we would want to allow the user to regroup her rules along dimensions orthogonal to the object types. And we call such groupings of rules agents, since they usually differ along dimensions like triggering events (*e.g.* when new mail comes in) and what kinds of objects to apply (*e.g.* apply to the objects in the "Urgent" folder).

  Incidentally, the rule sets in Lens also caused two problems in the user interface. First, users do not remember under which message type they

may have stored a particular rule. This is a direct consequence of forcing the user to group her rules along the message type hierarchy as opposed to along what and when she wants things done (*i.e.* the dimensions of the groupings using agents). Second, each rule in Lens has a Message type: field so that users can put rules in a general type (*e.g.* Message) and filter on message objects of more specialized types (*e.g.* NYT Article), without putting the rules in the rule set attached to the NYT Article type. Of course, this is a rather *ad hoc* solution to the first problem: it allows users to keep many rules in the Message rule set instead of spreading them out over several rule sets under different message types.

- "move" and "delete" marks to indicate the effects of rule actions. One example of such use is to allow users to create rules that depend on the actions of rules fired earlier. To do that, the rule actions need to leave behind a trail of side effects. In the case of Object Lens, there is no need to do this because a more physical model is used. Deleting a link really deletes it, and moving a link deletes it from the source and puts it in the destination.

POSSIBLE FEATURES FOR A FUTURE VERSION

- Good performance is crticial in hypertext systems.

While hypertext systems provide convenient ways of linking documents, they must also give the incentive for users to follow the chains of links if necessary. Otherwise, it defeats the very purpose of such systems if users find it hard to traverse the hypertext.

- Sorting links in folders by fields.

One feature missing in the tabular presentation of links of folders is to order them in ways prescribed by the user. For example, a folder containing links to ACTION REQUESTS may be required to be order according to the values of the Action Deadline: field.

- Browsing folder networks with automatic retrieval.

A useful feature to have is the ability to specify a few objects in a folder, and the links to be browsed. When the folder is displayed as a network, all objects that are linked to those already in the folder are also "pulled" into the folder, and they in turn "pull" others along.

- Backward reference.

The rule language we have now is limiting in that one cannot refer to specific information *in the objects world* when doing the matching. For instance, one cannot say things like "if this MEETING ANNOUNCEMENT has the same time *as an existing MEETING ANNOUNCEMENT* then ...". Backward reference is the ability to say "the <field name> of the <existing object>" *e.g.* the Time-of: (the reverse of the Time: field) field.

- Fisheye display of networks.

One of the problems of displaying links in a network is that it is difficut to find the link that one needs. A possible solution is the use of fisheye views [Fum16]. However, this can be rather inefficient. That is why we have, for a first pass, just introduce the Zoom in and Zoom out actions instead.

- Layout of the Objects on the Screen.

As more objects are incorporated into the objects world, it is necessary to find a way to present them on the screen without too much effort. At present, there are some features to lighten the burden, such as the Expand Link Object aspect and the automatic placement of objects of embedded links to the right of the embedding object. A more systematic scheme has to be found to cope with the increasing number of windows on the screen.

- Retaining information about descriptions used by senders.

As part of the Show History action, Object Lens can retain the descriptions used by senders, say, in the To: fields. That way, receivers can find out why they got their messages.

- Implementing the Anyone Server and, in general, remote agents.

This is discussed in the first paper in this thesis, and will not be further discussed here.

CONCLUSION

We presented a more general framework for the Lens system. Such a framework opens up very many possibilities in supporting cooperative work, information retrieval, and the automatic processing of information.

REFERENCES

[Con86]
Conklin, J., A Survey of Hypertext, MCC Technical Report, Number STP-356-86.

[Del87]
Delisle, N. M. and chwartz, M. D. Contexts – a Partitioning Concept for Hypertext, *ACM Transactions on Office Information Systems*, Volume 5, Number 2 (April 1987), 168-186.

[Fum86]
Fumas, G. W. Generalized Fisheye Views. *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, Boston, Massachusetts (April 1986), 16-22.

[Gar86]
Garrett, L. N., Smith, K. E., and Meyrowitz, N. Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System, *Proceedings of the Conference on Computer-Supported Cooperative Work*, MCC Software Technology Program, Austin, Texas (1986).

[Mal87a]
Malone, T. W., Grant, K. R., Turbak, F. A., Brobst, S. A., and Cohen, M. D. Intelligent Information Sharing Systems. *Communications of the ACM*, Volume 30, Number 5 (May 1987), 390-402.

[Mal87b]
Malone, T. W., Grant, K. R., Lai, K. Y., Rao, R., and Rosenblitt, D. Semistructured Messages are Surprisingly Useful for Computer-supported Coordination. *ACM Transactions on Office Information Systems*, Volume 5, Number 2 (April 1987), 115-131.

[Tri86]
Trigg, R., Suchman, L., and Halasz, F. Supporting Collaboratoin in NoteCards, *Conference on Computer-Supported Cooperative Work*, MCC Software Technology Program, Austin, Texas (1986).