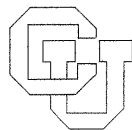


**Explorations in the Design of Intelligent Support Systems  
And Innovative User Interfaces \***

**Gerhard Fischer  
Anders Morch  
Charles Hair  
Andreas Lemke  
Berhard Bernstein  
Curt Stevens**

**CU-CS-388-88**



**University of Colorado at Boulder  
DEPARTMENT OF COMPUTER SCIENCE**

\* The research described in this report was supported by grants from Symbolics, Cambridge, MA, the Human Interface Program at MCC, Austin, TX, and the AT&T Foundation, New York, NY.

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS  
EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR(S) AND DO NOT  
NECESSARILY REFLECT THE VIEWS OF THE AGENCIES NAMED IN THE  
ACKNOWLEDGMENTS SECTION.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100

THE  
FIRST  
PART  
OF  
THE  
HISTORY  
OF  
THE  
CITY  
OF  
LONDON  
FROM  
THE  
BEGINNING  
TO  
THE  
PRESENT  
TIME  
BY  
JOHN  
STOW  
1597

# Explorations in the Design of Intelligent Support Systems and Innovative User Interfaces

Gerhard Fischer, Anders Morch, Charles Hair, Andreas Lemke,  
Bernard Bernstein, and Curt Stevens

Department of Computer Science and Institute of Cognitive Science  
University of Colorado, Boulder

March 1, 1988

Our goal is to establish, both by theoretical work and by building prototypical systems, the scientific foundations for the construction of intelligent systems which serve as amplifiers of human capabilities (e.g., to expand human memory, augment human reasoning, and facilitate human communication). A prerequisite for intelligent systems is that we understand the information processing possibilities and limitations of the human *and* the computer. We apply basic, qualitative theories of human thinking to guide the design of innovative systems. Our systems should not only be significant as technical achievements in computer science, but also because they are based upon principled analyses of how one can best help people to cope with complex information systems. Working in intelligent systems, it is not sufficient to know how to build these systems; one must discover which systems are worth building.

Knowledge-Based Systems (KBS) and Human-Computer Communication (HCC) are two crucial research areas for these goals. We are especially interested to understand the possibilities of pursuing these two research areas together. The rationale for this approach is that on the one hand effective human-computer communication is more than creating attractive displays on a CRT screen: it requires providing the computer with a considerable body of relevant world knowledge as well as knowledge about the psychological characteristics and understanding of the user; on the other hand, the use of knowledge-based systems and expert systems will be severely limited if we are unable to eliminate the communication bottleneck.

This report describes a number of explorations in this research area. It contains the following parts:

A. A paper by Gerhard Fischer and Anders Morch entitled "**CRACK: A Critiquing Approach to Cooperative Kitchen Design**" which explores the idea of human problem-domain communication and cooperative problem solving in order to allow users, who are not computer experts, to use computers for their own purposes. CRACK is an operational system which runs on the Symbolics using the ART system from inference cooperation. For interested readers, a videotape presentation is available from our research group.

B. A paper by Charles Hair and Andreas C. Lemke entitled "**FRAMER: a Design Environment for the Construction of Window-Based Interfaces**". FRAMER is an extension to the FRAME-UP system developed by Symbolics which incorporates a menu-driven interface, support for design by modification and a critic. FRAMER is an operational prototype running on the Symbolics.

C. A paper by Bernard Bernstein and Curt Stevens entitled "**NEWSCOPE: Towards a User Modeled**

**Personal Information Retrieval System"**. NEWSCOPE is an user interface to the News System which incorporates innovative user interface techniques to cope with the huge information volume delivered within the News system. NEWSCOPE is an operational prototype running on the Symbolics.

D. A paper by Curt Stevens and Andreas C. Lemke entitled "**Experiences with the Genera User Interface Facilities**". The FRAMER and the NEWSCOPE system are built using the Genera 7 Symbolics programming environment taking advantage of the presentation substrate. This is a brief, critical evaluation from a user perspective.

### **Acknowledgements**

The research described in this report was supported by grants from Symbolics, Cambridge, MA, the Human Interface Program at MCC, Austin, TX and the AT&T Foundation, New York, NY.

# CRACK: A Critiquing Approach to Cooperative Kitchen Design

Gerhard Fischer and Anders Morch  
Department of Computer Science and Institute of Cognitive Science  
University of Colorado, Boulder

## Abstract:

Human problem-domain communication and cooperative problem solving are two enabling conditions that allow users, who are not computer experts, to use computers for their own purposes. Computer-based critics, a specific class of intelligent support systems, are most effective if they are embedded in a framework defined by human problem-domain communication and cooperative problem solving.

CRACK is a specific critic system which supports users designing kitchens. It provides a set of domain specific building blocks and has knowledge about how to combine these building blocks into useful designs. It uses this knowledge "to look over the shoulder" of a user carrying out a specific design. If CRACK, based on its understanding of kitchen design, discovers a shortcoming in users' designs, it offers criticism, suggestions, and explanations and assists users in improving their designs through a cooperative problem solving process. CRACK is not an expert system that dominates the design process by generating new designs from high-level goals or resolving design conflicts automatically. Users control the behavior of the system at all times (e.g., the critiquing can be "turned on and off"), and if users disagree with CRACK, they can modify its knowledge base.

## Keywords:

knowledge-based computer-aided design, critics, cooperative problem solving, intelligent support systems, human problem-domain communication, kitchen design



## Table of Contents

<b>1. Introduction</b>	<b>2</b>
<b>2. Cooperative Problem Solving Systems</b>	<b>2</b>
2.1 The Critiquing Approach in Human-Computer Communication	2
2.2 Human Problem-Domain Communication	3
2.3 Cooperative Problem Solving	4
2.4 The Role of Critics in Cooperative Problem Solving Systems	6
<b>3. CRACK: A Critic for Kitchen Design</b>	<b>7</b>
3.1 The Problem Domain: Kitchen Design	7
3.2 Knowledge Acquisition	7
3.3 A User Interface Based on the World Model	9
3.4 The Critics	9
3.5 Explanations	10
3.6 Modification of the Design Knowledge	10
<b>4. Evaluation</b>	<b>13</b>
<b>5. Extensions</b>	<b>14</b>

## List of Figures

<b>Figure 2-1: Basic Components to Support Cooperative Problem Solving Processes</b>	<b>5</b>
<b>Figure 3-1: Suggestions from the SINK-CRITIC</b>	<b>8</b>
<b>Figure 3-2: Design Unit Hierarchy</b>	<b>10</b>
<b>Figure 3-3: The Work-Triangle Critic -- A Critic for a Higher Level Concept</b>	<b>11</b>
<b>Figure 3-4: Explanation for AWAY-FROM Relation in RANGE-CRITIC</b>	<b>12</b>
<b>Figure 3-5: Modifying Design Knowledge</b>	<b>13</b>



## 1. Introduction

Many aspects of human-computer systems have not kept pace with the dramatic progress in hardware development. One of the major challenges is to enable occasional users, who are experts in some application domain, to take advantage of the available computational power and *to use the computer for a purpose chosen by themselves* [Illich 73]. Most computer users feel that computer systems are unfriendly, not cooperative, and that it takes too much time and too much effort to get something done. They feel that they are dependent on specialists and notice that "software is not soft" (i.e., the behavior of a system can not be changed without a major reprogramming of it).

In this paper we describe a framework to overcome these limitations with the help of knowledge-based systems, qualitatively different human-computer communication, and the use of the computer for educational and training purposes in which the users are in control of the communication process. We illustrate our general approach with a detailed discussion of CRACK, a critic for kitchen design. An objective of CRACK is to blend the designer and the computer into a problem solving team to produce cooperatively better designs than each of them working alone. CRACK is capable of critiquing users, providing suggestions and explanations, and allowing users to change the behavior of the system. An evaluation of the current version of CRACK will be given and current limitations and future enhancements discussed.

## 2. Cooperative Problem Solving Systems

### 2.1 The Critiquing Approach in Human-Computer Communication

Three major communication paradigms in human-computer systems are: tutoring, consultation, and critiquing.

*Tutoring* (e.g., as in the LSP-TUTOR [Anderson et al. 84; Anderson, Reiser 85] and in the PROUST system [Johnson, Soloway 84]) provides an appropriate framework for getting started to learn a new system. In tutoring systems, one can predesign a sequence of microworlds [Burton, Brown, Fischer 84] and lead a user through them. However, tutoring offers little help in supporting users in situations where they are involved in their "own doing". Tutoring is not task-driven because the total set of tasks *cannot* be anticipated. Instead, the system controls the dialogue, and the user has little control over what to do next.

*Consultation* is a frequently used interaction model in expert systems [Buchanan, Shortliffe 84]. From a system designer's point of view, this model has the advantage of being clear and simple: the program controls the dialogue (in much the same way as a human consultant does) by asking for specific items of data about the problem at hand. The disadvantages are that it prevents a user from volunteering information [Fischer, Stevens 87], and it does not support mixed-initiative dialogues.

The *critiquing model* allows users to pursue their own goals, and the program interrupts only if the user's behavior is judged to be significantly inferior to what the program would have done. It is based on empirical observations [Carroll, McKendree 87] that users are often unwilling to learn more about a system or a tool than is necessary for the immediate solution of their current problem. To be able to successfully cope with new problems as they arise, a critic is required that generates advice tailored to the specific needs of the users. The critiquing approach provides information only when it becomes

relevant. It eliminates the burden of learning new things in neutral settings when the user does not know whether the information will ever be used and has difficulty imagining an application.

We have developed programs which instantiate a number of different aspects of the critiquing model. The active help system ACTIVIST [Fischer, Lemke, Schwab 85] looks a user (working with an editor) "over the shoulder" and infers from user actions the plan which the user wants to pursue and compares it with its own plan. Information about the user's behavior is stored in the model of the user. A separate tutoring module (taking the information in the model of the user into account) decides when to offer help and advice. The LISP-CRITIC [Fischer 87a] enhances incremental learning of LISP and supports learning strategies such as learning on demand (i.e., information is provided when needed). It has knowledge about how to improve LISP programs locally, following a style defined by its rules. The system operates by using a large set of transformation rules which describe how to improve Lisp code. The user's code is matched against the rules' premises, and the transformations suggested are given to the user. Additional tools are available to explain and illustrate the advice.

A number of issues have been learned constructing these systems. Criticism and volunteered advice is most welcome when it is directly relevant to the problem or the task the user is working on. The major problem in systems of this kind is not to make them speak up but to keep them quiet most of the time. To achieve this requires elaborate knowledge structures (e.g., models of the users and tutorial strategies). In addition, users must be put in control of the communication with the system in order to be able to ignore irrelevant volunteered information (they may already know it or they may regard it as not relevant) and to turn the critic off if they want to be left alone.

## 2.2 Human Problem-Domain Communication

Most computer users are not interested in computers per se, but rather want to use the computer to solve problems and to accomplish certain tasks. To shape the computer into a truly usable and useful medium, we have to make it invisible as a tool and let users work *directly* on their problems and tasks.

Human problem-domain communication [Fischer, Lemke 88] provides a new level of quality in human-computer communication because the important objects and abstract operations of a given application domain are built directly into the computer. This implies that the user can operate with personally meaningful abstractions. In most cases it is not desirable to eliminate the semantics of a problem domain by reducing the information to formulas in first-order logic or to general graphs. *Systematic domains* [Winograd, Flores 86], defining the major abstractions of a problem domain and their interrelationships, are needed to support human problem-domain communication.

### Construction Kits

Construction kits are system components that represent steps towards human problem-domain communication by providing a set of building blocks that model a problem domain. The building blocks define a design space (the set of all possible designs that can be created by combining these blocks) and a design vocabulary<sup>1</sup>. Construction kits can be seen as domain specific programming languages which help users to formulate solutions to complex problems and to create complex environments without

---

<sup>1</sup>The specific design vocabulary for CRACK is represented as a set of icons in a palette (see Figure 3-1).

having to master the many details of programming inherent in general programming languages. They offer the potential advantage of eliminating a number of prerequisite skills, thus allowing users much more time to practice and work in their actual area of interest.

The PinBall and Music Construction Kits (two interesting programs for the Macintosh from Electronic Arts [Fischer, Lemke 88]) provide domain-specific building blocks (bumpers, flippers; staves, piano keyboard, notes, sharps, etc.) to build artifacts in the two domains of pinball machines and musical composition. Users can interact with these systems in terms with which they are already familiar, and they need not learn abstractions peculiar to a particular computer system.

Our empirical investigations have shown that these systems come close (within their scope) to our notion of human problem-domain communication. Users familiar with the problem domains but inexperienced with computers had few problems using these systems, whereas computer experts unfamiliar with the problem domains were unable to exploit the power of these systems. Persons using these systems are designing artifacts, without the need for programming by writing statements in a programming language. Our subjects had a sense of accomplishment in using these construction kits because they enabled them to construct something quickly.

In the context of this paper, individual building blocks will be referred to as *design units*. Eastman [Eastman 69] defines a design unit (DU) as a physical element that can be selected and manipulated during the design process. DUs can further be organized into hierarchies which arrange them according to the physical elements of which they are a part.

### The Limitations of Construction Kits

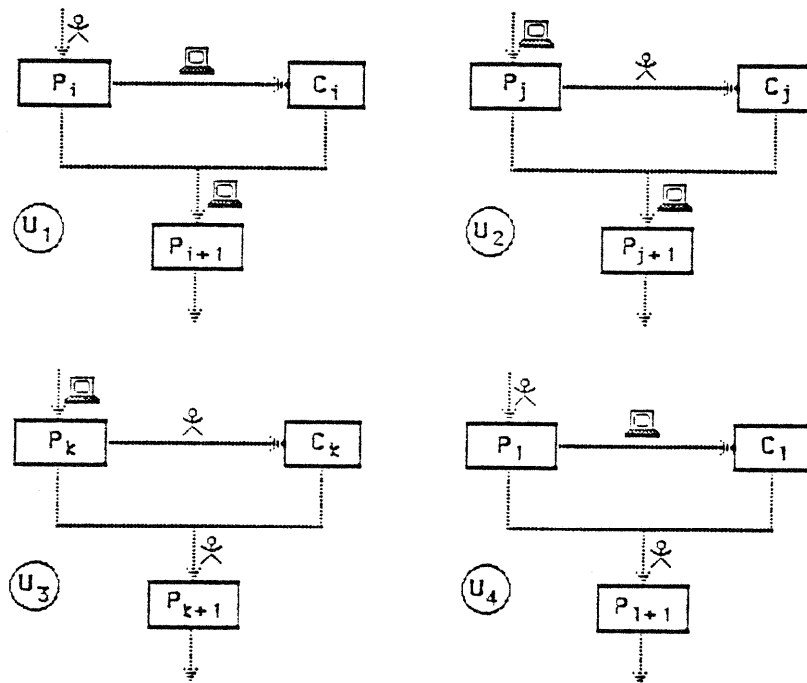
Evaluating the Pinball and Music Construction Kits as prototypical examples against our objective of enhancing human problem-domain communication, we have found that their major shortcoming is that they do not assist the user in constructing interesting and useful artifacts in the application domain. The Pinball Construction Kit allows users to build games in which balls get stuck in certain corners and certain devices can never be reached [Hutchins, Hollan, Norman 86]. The insufficiency of just providing design units in CRACK can be characterized by the fact that "kitchen design is more than providing a number of appliances". Design environments [Fischer, Lemke 88] are needed that assist users in constructing truly interesting artifacts. The primitives of a programming language or the elements of a construction kit give little guidance on how to construct a complex artifact which achieves a certain purpose. Design critics go beyond construction kits in that they bring to bear general knowledge about design (e.g., which meaningful artifacts can be constructed, how and which design units can be combined with each other) that is useful for the designer.

### 2.3 Cooperative Problem Solving

The intelligent support systems, which we have constructed so far (e.g., [Fischer 87a; Fischer, Lemke, Schwab 85]), are "one-shot" affairs. They may give criticism and advice, but the information provided by them does not serve as a starting point for a cooperative problem-solving process. Human advisory dialogues [Carroll, McKendree 87] are judged successful when they allow a shared control of the dialogue. We have explored the issues associated with shared control in a system architecture which allows the volunteering of advice by the user [Fischer, Stevens 87]. When humans (e.g., a novice and an

expert) communicate, much more goes on than just the request for factual information. Novices may not be able to articulate their questions without the help of the expert. The criticism or advice given by the expert may not be understood, and/or the advisee may request an explanation of it. Experts sometimes have difficulties seeing the problem from the novices' point of view. Each communication partner may hypothesize that the other partner misunderstood him/her, or they may provide information for which they were not explicitly asked. The criticism provided in such interactions can serve multiple purposes: it can become itself an object of interrogation, and it can serve as a starting point for a learning process [Fischer 87a].

Cooperative problem-solving processes can be modeled using the basic primitives  $U_1$  to  $U_4$  represented in Figure 2-1. The four primitives can be combined in arbitrary ways. CRACK in its current form supports  $U_4$ . To capture  $U_1$ , CRACK has to be extended such that it can solve certain problems by itself. This can be done by associating local expert system modules with each design unit.



$P_i$ : product version  $i$   
 $C_i$ : criticism  $i$

Figure 2-1: Basic Components to Support Cooperative Problem Solving Processes

$U_1$  through  $U_4$  are the four possible units of cooperative problem solving processes. Either the human or the computer can criticize a product that was generated by either of them. One of them then creates a new product based on the previous version and the criticism.

Actions in cooperative problem-solving systems should not cause unresolvable breakdowns of the inter-

action and should not be regarded as errors, but should be an integral part of the process of accomplishing a task. All efforts in a cooperative problem-solving process should be regarded as iterations towards a goal. Misunderstandings should lead to a situation which can be described as "Let's talk about it" [Lewis, Norman 86]. The goal of a cooperative endeavor is neither to find fault nor to assess blame, but rather to get the task done.

It is insufficient for intelligent support systems just to solve a problem or to provide information. They need to do this in a way that the user can understand and question their criticism. It is one of our working assumptions that learners and practitioners will not ask a computer program for advice if they have to treat the program as an unexaminable source of expertise. One has to provide windows into the knowledge base and into the reasoning processes of these systems at a level which is understandable by the user. The users should be able to query the computer for suggestions and explanations, and they should be able to modify and augment the knowledge of the critic if they are dissatisfied with the information received.

## 2.4 The Role of Critics in Cooperative Problem Solving Systems

Design can be viewed as problem solving where complex artifacts are constructed from simple building blocks in order to find a *satisficing* solution to a design problem. Simon [Simon 81] defines satisficing as a means to look for adequate or satisfactorily solutions rather than optimal ones. In the same way as construction kits constrain the design space by limiting the number of design units a user can select, critics constrain the design space by making the user aware of the distinction between satisficing and non-satisficing arrangements of design units. Critics are needed to guide users in unfamiliar problem domains. Critics in CRACK are procedures for detecting non-satisficing partial designs and can be classified along the following dimensions:

**Activation.** Critics can be *active* and activate themselves when they detect a non-satisficing arrangement of design units, or they can be *passive* and the user has to ask for an evaluation. An active critic can be envisioned as a knowledgeable human designer watching over a user's shoulder and critiquing each time an arrangement is detected that violates his or her notion about an appropriate solution. For example in kitchen design this can be complaints in the form of: "*sink not in front of a window*" or "*refrigerator next to the range*". This type of criticism will make the users aware of their non-satisficing design at an early point which makes it easier for them to correct it, but at the same time they might find it a nuisance to have someone continuously critique them and not give them any chance to develop something of their own for some period of time. A passive critic does not have this problem since the users themselves request an evaluation when they have completed a partial design. Active critics seem to be suited to guide novice users, and passive, user-initiated critics seem to be more appropriate for intermediate users.

**Positiveness.** Critics can either be positive (praising superior design) or negative (complaining about inferior, non-satisficing design). Real life critics (art critics, movie critics) are both positive and negative.

**Granularity.** The grain size of critics determines whether they are oriented towards local aspects of a partial design or a global perspective of the total design. A *sink critic* is an example of a local critic since it is only concerned about the low-level design unit "sink". A *work triangle critic* is concerned with a larger portion of the design since it is associated with the work triangle<sup>2</sup> which is an abstraction of several

---

<sup>2</sup>The *work triangle* is the center front distance between the three appliances sink, range and refrigerator.

appliances. A *kitchen critic* which is concerned about the kitchen's balance and total look is an example of a global critic.

### 3. CRACK: A Critic for Kitchen Design

#### 3.1 The Problem Domain: Kitchen Design

CRACK is a kitchen design critic which aids users in designing a kitchen floor plan layout while sitting in front of a graphics workstation (see Figure 3-1).

Ill-defined problem areas where satisficing rather than optimizing is the goal are well suited for the critiquing approach. Kitchen design (as an area of architectural design) is still an ill-defined problem despite the existence of some well-established design principles. Architectural design is characterized by having no strong theoretical basis as compared to other design areas such as structural engineering and computer design, and architects are not trying to find optimal solutions to design problems but rather tradeoffs within a solution space bounded by external constraints. CRACK's critiquing approach to design is directed towards detecting non-satisficing partial solutions.

#### 3.2 Knowledge Acquisition

Domain dependent design knowledge represented in CRACK has been acquired from kitchen design books and from professional kitchen designers whose knowledge was captured by means of protocol analysis and a questionnaire.

**Kitchen Design Books.** Our initial exposure to the standards of American kitchens was from the series of texts compiled by the *Small Homes Council-Building Research Council* at the University of Illinois. The most useful manual was *Kitchen Planning Principles - Equipment - Appliances* [Jones, Kapple 84], but also the kitchen design book [Paradies 73] provided insightful information. Most of the design parameters used in design units and explanations for critics are taken from these two texts.

**Protocol Analysis.** Two professional kitchen designers cooperated with us in this research. *Protocol analysis* [Ericsson, Simon 84] was used to gather a set of protocols. The two professionals were given typical scenarios which included a sample floor plan and a hypothetical client providing needs and desires. They were asked to plan a kitchen for this client in the space provided. In order to capture all the steps involved, including the ones which designers normally do not communicate, they were asked to *think aloud* during the design process. If they still made some "big jumps" in the reasoning process, which often happened, they were interrupted, and the experimenter asked questions to bridge these intermediate gaps. The sessions were recorded, and four protocols were gathered and analyzed.

The protocol studies revealed *domain related concepts* specific to kitchen design. Spatial relationships such as *in front of*, *next to* and *near* have their own meaning in this domain. *In front of* is used to refer to a relation between an equipment (appliance or cabinet) and a wall fixture (door, window, plumbing), e.g., sink *in-front-of* window. *Next to* refers to two appliances which are side by side along a wall assembly, e.g., sink *next-to* dishwasher. *Near* refers to equipment which is not immediately next to each other, but still within reach, meaning about 4-8 feet apart, e.g., sink *near* refrigerator.

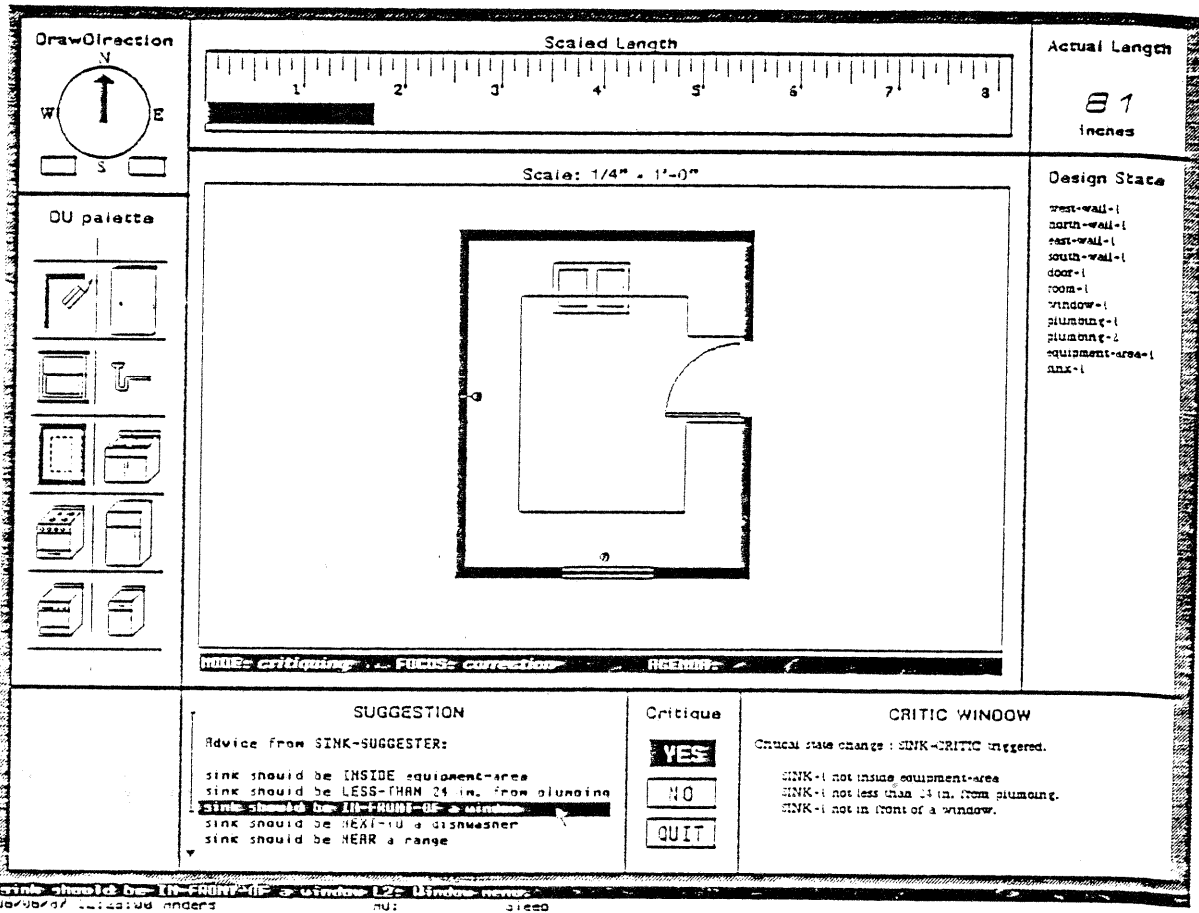


Figure 3-1: Suggestions from the SINK-CRITIC

CRACK's user interface is based on the world model and the metaphor of an "architect's workbench". Design units are selected from the DU Palette, and corresponding architectural symbols are moved around in the work area (the center window). Operations on DUs are initiated by clicking on their instance name in the Design State window. Suggestions, criticism and operations can be questioned by clicking on the text. Compass, ruler and actual length are active values used during wall drawing and door/window positioning to support the user with graphical data. Critiquing can be turned on and off.

**Questionnaire.** The protocol studies were useful in understanding the design process, which includes in *what order* the various design units are applied and *how to* select their type and properties such as width and depth. For the computer implementation, more concrete information in the form of specific values for design parameters was needed. Some of these values were found in the books mentioned above, but most of them were obtained by asking the designers to fill out a questionnaire.

### 3.3 A User Interface Based on the World Model

CRACK's user interface is based on the *world model* metaphor [Hutchins, Hollan, Norman 86]. Users can directly manipulate the objects in the world of kitchen design. A direct manipulation interaction style using the mouse and context-sensitive menus makes it easy to learn CRACK. The interface tries to model an "architect's workbench" which is a familiar environment for designers. Architectural tools such as pencil, paper, ruler and compass are part of the graphics interface. Users can "draw" the walls of the room with pencil and ruler, and they can select standard kitchen appliances (sink, range, refrigerator, etc.) from a design unit palette and move them around with the mouse to desired locations. The user interface of CRACK allows users to engage themselves directly in their application and it is a step towards *human problem-domain communication* as described in section 2.2.

### 3.4 The Critics

The critics in CRACK are rules which are activated after each state change and they send information to the user when non-satisficing partial solutions are detected. State changes are all instance creations of design units and any design unit manipulation (e.g., *move*, *rotate*, *scale*). A non-satisficing solution is an arrangement of design units which violates one or more of the relations between them. These relations are based on design knowledge acquired by the methods described in section 3.2, but can be modified by a user (see section 3.6).

The critics in CRACK are *negative* in the sense that they are only complaining about non-satisficing configurations instead of also praising especially useful or interesting configurations. A typical critic is *SINK-1 not in-front-of a window* (see Figure 3-1). This is a complaint about the current screen state after a critical state change caused by *SINK-1*.

The grain size of critics are determined by the design units (DU). Each DU has an associated critic. For example the DU *sink* has the critic *SINK-CRITIC*. The DUs have no knowledge about themselves except for their screen position and their location in the DU hierarchy (Figure 3-2). The knowledge about a DU's relations with other DUs is represented by its critic. Not all critics are related to low-level DUs like the sink. The *WORK-TRIANGLE-CRITIC* tests to see if the center front distance between the appliances sink, range, and refrigerator is less than 23 feet (see Figure 3-3).

A critic consists of a set of geometrical relations which can either be true or false. For example in the *SINK-CRITIC* some relations in prefix notation are: (*INSIDE sink equipment-area*), (*IN-FRONT-OF sink window*), (*NEXT-TO sink dishwasher*), (*LESS-THAN sink plumbing 24*) and (*NEAR sink refrigerator*). These are some of the relations checked each time the *SINK-CRITIC* is triggered, and complaints in the form of: *SINK-1 not in-front-of window*, *SINK-1 not less-than 24 inches from plumbing*, etc., are printed out to a critic-window on the screen in cases where these relations are violated (see Figure 3-1).

The actual geometrical comparisons are performed by actions<sup>3</sup> defined on a pair of design units. For

---

<sup>3</sup>CRACK is implemented using ART, a knowledge-based development environment from Inference Corporation that runs on a SYMBOLICS Lisp machine. "Action" is the ART terminology for a method defined on objects or slots in an object-oriented programming language.



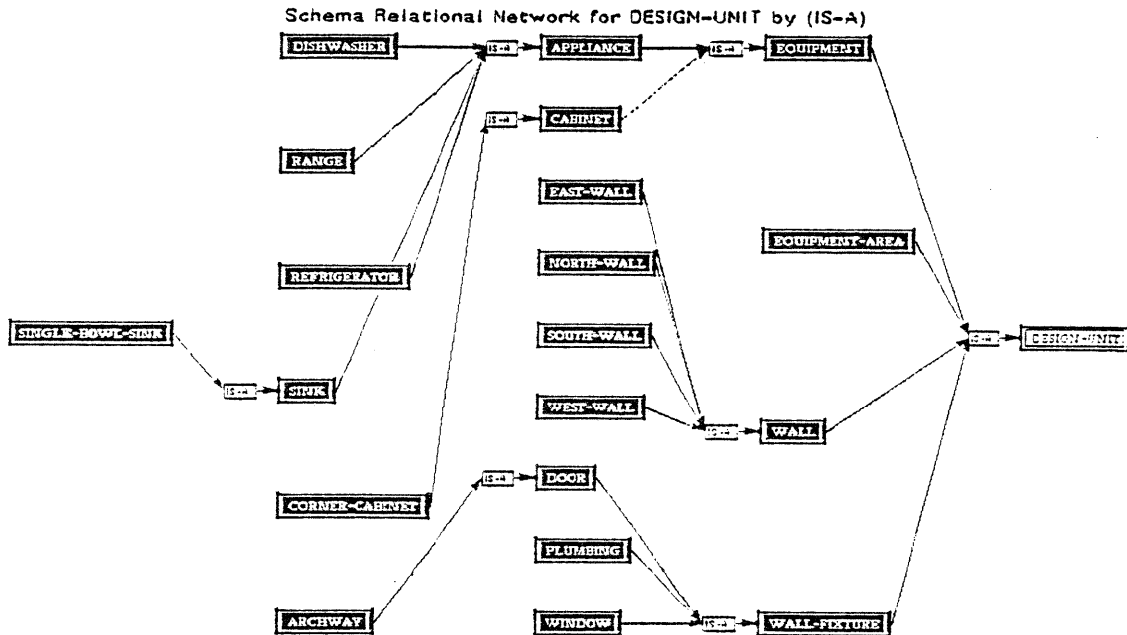


Figure 3-2: Design Unit Hierarchy

example (defaction in-front-of equipment wall-fixture () ...) defines an action on two generic design units equipment and wall-fixture (see Figure 3-2). All pairs of DUs which inherit from these will also have the method IN-FRONT-OF defined. For example (IN-FRONT-OF sink window), (IN-FRONT-OF range door), (IN-FRONT-OF cabinet plumbing), (IN-FRONT-OF appliance door) are all legal ways to invoke (send a message to) the IN-FRONT-OF method. This way of interchanging DUs in relations will be used to facilitate critic modifications.

### 3.5 Explanations

A user can ask for an explanation of each relation belonging to a critic (see Figure 3-4). For example a user can ask why a range should be AWAY-FROM a window, and an appropriate answer will be given: *You put yourself in danger if trying to open the window while the range is on, and there is a substantial fire hazard if flammable curtains are installed.* These explanations are "hard-wired" into the system in order to explain the design knowledge in kitchen design and cannot be modified by a user in the current implementation of CRACK.

### 3.6 Modification of the Design Knowledge

CRACK allows the user to control the firing of critics at three levels: all critiquing can be turned on or off, individual critics can be enabled or disabled, and specific relations in a critic can be modified. When critiquing is turned off (which it is by default), CRACK acts like a construction kit without any design

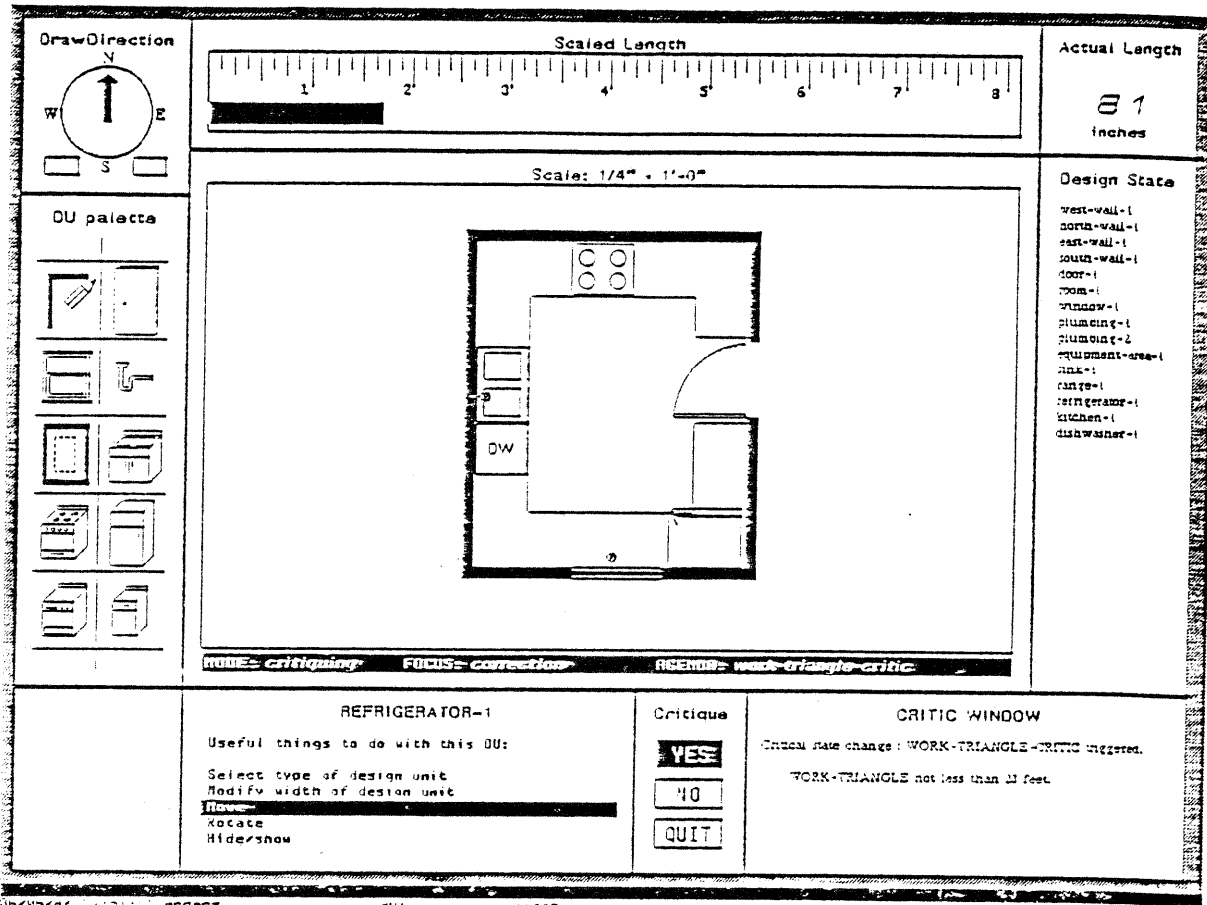
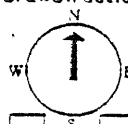
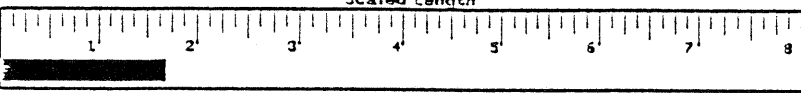
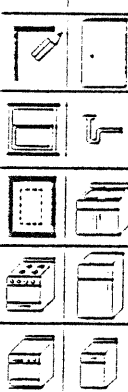
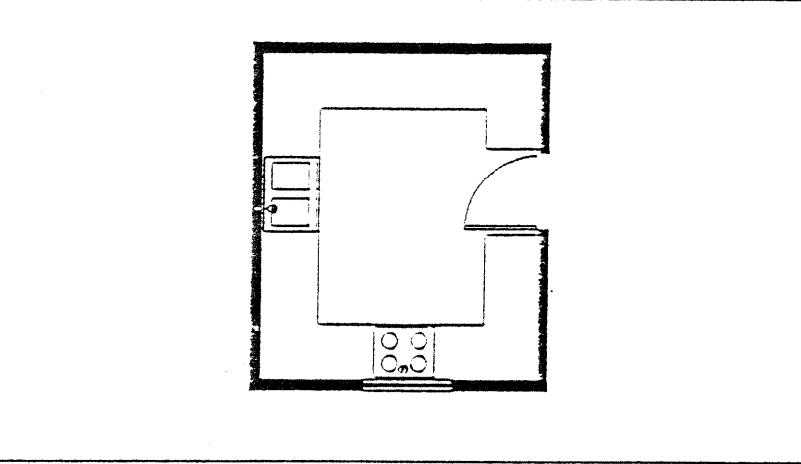


Figure 3-3: The Work-Triangle Critic -- A Critic for a Higher Level Concept

knowledge to guide users, just like the Pinball Construction Kit. When critiquing is enabled, all critics are active. An individual critic can be disabled if a user does not like its criticism, or if its knowledge has been acquired by a user and is not needed any more. By default, all critics are enabled.

CRACK allows users to modify critics -- an important requirement for cooperative problem solving systems. This modification can either take the form of a replacement or a removal of one of the relations. The relation `IN-FRONT-OF` can be replaced by either: `NOT-IN-FRONT-OF`, `CLOSE-TO`, `NOT-CLOSE-TO`, or `NO-RELATION`. `NO-RELATION` means no relationship between these two DUs. When a critic is modified, CRACK's suggestions for this DU are updated to reflect the new understanding of the problem. In this way a user who does not want to have `SINK-1` `in-front` of a `window` can replace this relation with `SINK-1` `close-to` `window` or no relation at all between `SINK-1` and `window`. Modified in this way, CRACK will not critique the user any more for not putting `SINK-1` in front of the `window`. The actual modification is done by having rules that modify and recompile other rules during run-time, as seen in the command window in Figure 3-5.

DrawDirection 	Scaled Length 		Actual Length 81 inches
DU palette 	Scale: 1/4" = 1'-0" 		Design State west-wall-1 north-wall-1 east-wall-1 south-wall-1 door-1 room-1 window-1 plumbing-1 plumbing-2 equipment-area-1 sink-1 range-1
EXPLANATION Putting the range in front of a window is not recommended for several reasons. Firstly you put yourself in danger by opening a window while heating elements are on. Furthermore if flammable curtains are installed, the potential fire hazard is substantial.		Critique <input checked="" type="button" value="YES"/> <input type="button" value="NO"/> <input type="button" value="QUIT"/>	CRITIC WINDOW Critical state change: RANGE-CRITIC triggered. <del>RANGE-CRITIC NOT TRIGGERED AWAY FROM WINDOW</del>

Home-RC flag-from relation menu  
 version/ selected orders      AU:      31220

Figure 3-4: Explanation for AWAY-FROM Relation in RANGE-CRITIC

This feature allows CRACK to *learn* to see the problem from a new perspective in order to better guide users towards their goal. For example, the "metarule" MODIFY-SINK-CRITIC redefines the rule SINK-CRITIC with the new relation a user has selected for substitution. Next time SINK-CRITIC is triggered or a suggestion from SINK-SUGGESTER is requested, this correction will be in effect. This modification will be permanent until another user modifies the same relation again, and it supports cooperative problem solving since both user and computer are critiquing and correcting each other in order to achieve a common goal. A DOMAIN-CRITIC is fired each time before a critic is redefined to warn users about the fact that they are modifying permanently stored domain knowledge. An UNDO is available if the modification needs to be revoked.

The screenshot displays the CRACK software interface. At the top, there is a 'DrawDirection' compass showing North (N), South (S), East (E), and West (W). To its right is a 'Scaled Length' ruler from 0 to 8 inches. Further right, the 'Actual Length' is shown as 81 inches. Below the ruler is a 'Scale: 1/4" = 1'-0"' indicator. The central 'Design State' window shows a kitchen floor plan with various elements labeled: west-wall-1, north-wall-1, east-wall-1, south-wall-1, door-1, room-1, window-1, plumbing-1, plumbing-2, equipment-area-1, and sink-1. To the left of the design window is a 'DU palette' with icons for different kitchen fixtures. Below the design window is a status bar with the text: `NAME: critiquing - FOCUS: modification - RECEIVE: modify-sink-critic`. At the bottom, there are three windows: 'COMMAND WINDOW' showing a terminal prompt and a warning message about redefining a rule; 'Critique' with buttons for 'YES', 'NO', and 'QUIT'; and 'CRITIC WINDOW' showing a message about a domain change.

Figure 3-5: Modifying Design Knowledge

## 4. Evaluation

CRACK has been an operational system for several months and we have accumulated some feedback about its strength and shortcomings. One of our colleagues who (as a non-professional kitchen designer) had just remodeled his kitchen considered the use of CRACK an important experience. The criticism that the system generated during his design process illustrated several design concepts which he was not aware of at the time of the remodeling. In addition to being able to generate a specific design for a kitchen, our colleague increased his general knowledge about kitchen design.

The system was also used by a design methodologist who considered the cooperative, user-dominated approach of CRACK its most important feature. He felt it was this feature which sets CRACK truly apart from expert system oriented design tools where users have little control and are often reduced to spectators of

the system's operations. In the current version of CRACK, we have deliberately not concentrated our efforts on equipping the system with its own design capabilities. One may also ask why critics, if they are in principle able to solve a problem, do not just do it themselves. The rationale is that users increase their knowledge and their independence by working with systems that do not do the work for them, but make the arrangements necessary for them to do it themselves. Too much assistance and too many automatic procedures can reduce the users' motivation due to lack of challenge.

In comparison to most current CAD systems which are merely drafting tools rather than design tools, CRACK has some "understanding" of the design space. This knowledge allows the system to critique a design during the design process -- a capability absent in CAD systems.

We have developed critic systems in a number of areas (e.g., the LISP-CRITIC and CRACK), emphasizing different issues (e.g. level of analysis, narrowly bounded problem domain versus open problem domain, active versus passive, etc). We expect that by a careful analysis and detailed comparison of these system building efforts, we will be able to develop general design principles which will support the design of critics and intelligent support system in other domains.

## 5. Extensions

As the brief discussion in the last section indicated, CRACK in its current form is a useful and usable system. But the general framework (i.e., human problem-domain communication and cooperative problem solving) on which CRACK is based and our previous research suggest a number of future enhancements.

**Design by Redesign.** Instead of starting design with basic building blocks, prototypical solutions that can be manipulated and refined through redesign [Fischer 87b] are important enrichments for designers and enlarge their design possibilities. *Model kitchens* could be stored within CRACK and adequate support tools to find, inspect, and modify these prototypical solutions could be provided.

**Higher-level Concepts.** Currently, all critics in CRACK (except the ~~WORK-TRIANGLE-CRITIC~~) are associated with low-level equipment DUs (sink, range, refrigerator, etc.). Our protocol studies (see section 3.2) clearly indicated that kitchen designers use higher level concepts. These higher level concepts also require critics, e.g., a ~~KITCHEN-CRITIC~~ that tests for global concepts such as: at least 72 inches of counter space, maximize cabinet storage, minimize cost, and the total look of the kitchen.

**Support for the Preferences of Individual Users.** For users with special demands and desires, context-sensitive critics are needed which are tailored to individual preferences. The current approach in CRACK is limited to critiquing the ideal user designing a standard kitchen. Explicit user models need to be incorporated into critics to serve individual users better.

**More Guidance with Graphical Support.** Users of CRACK could be advised where (according to the system's understanding) a design unit selected from the palette could be placed. The system could highlight these areas. The integration of this feature into the system would have to be carefully evaluated, because it would provide substantially more guidance, thereby reducing the opportunities for the users to explore designs by themselves.

**Deliberation.** Users can modify design knowledge in CRACK, changing the behavior of the system per-

manently (see section 3.6). But this operation deletes the previously stored knowledge. In future versions of CRACK, we will support the concept of *deliberation* by which an arbitrary number of arguments (support, refutation, including associated explanations) can be stored in the system's knowledge base, representing the views of different designers. With this capability, the different styles and strategies of a number of designers can be represented, inspected and selected as the basis for critiquing. The knowledge base of CRACK could evolve by having designers use the system to integrate their expertise by adding new rules to the system. This feature would acknowledge that expertise in design is never complete and highly controversial and would allow learners to acquaint themselves with different design philosophies.

## References

- [Anderson et al. 84]  
J.R. Anderson, C.F. Boyle, R.G. Farrell, B.J. Reiser, *Cognitive Principles in the Design of Computer Tutors*, Proceedings of the Sixth Annual Conference of the Cognitive Science Society, Boulder, CO, June 1984, pp. 2-9.
- [Anderson, Reiser 85]  
J.R. Anderson, B.J. Reiser, *The LISP Tutor*, BYTE, Vol. 10, No. 4, April 1985, pp. 159-175.
- [Buchanan, Shortliffe 84]  
B.G. Buchanan, E.H. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley Publishing Company, Reading, MA, 1984.
- [Burton, Brown, Fischer 84]  
R.R. Burton, J.S. Brown, G. Fischer, *Analysis of Skiing as a Success Model of Instruction: Manipulating the Learning Environment to Enhance Skill Acquisition*, in B. Rogoff, J. Lave (eds.), *Everyday Cognition: Its Development in Social Context*, Harvard University Press, Cambridge, MA - London, 1984, pp. 139-150.
- [Carroll, McKendree 87]  
J.M. Carroll, J. McKendree, *Interface Design Issues for Advice-Giving Expert Systems*, Communications of the ACM, Vol. 30, No. 1, January 1987, pp. 14-31.
- [Eastman 69]  
C.M. Eastman, *Cognitive Processes and Ill-Defined Problems: A Case Study from Design*, Proceedings of the International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, May 1969, pp. 669-675.
- [Ericsson, Simon 84]  
K.A. Ericsson, H.A. Simon, *Protocol Analysis: Verbal Reports as Data*, The MIT Press, Cambridge, MA, 1984.
- [Fischer 87a]  
G. Fischer, *A Critic for LISP*, Proceedings of the 10th International Joint Conference on Artificial Intelligence (Milan, Italy), J. McDermott (ed.), Morgan Kaufmann Publishers, Los Altos, CA, August 1987, pp. 177-184.
- [Fischer 87b]  
G. Fischer, *Cognitive View of Reuse and Redesign*, IEEE Software, Special Issue on Reusability, July 1987, pp. 60-72.
- [Fischer, Lemke 88]  
G. Fischer, A.C. Lemke, *Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication*, Human-Computer Interaction, Vol. 3, No. 2, 1988.
- [Fischer, Lemke, Schwab 85]  
G. Fischer, A.C. Lemke, T. Schwab, *Knowledge-Based Help Systems*, Human Factors in Computing Systems, CHI'85 Conference Proceedings (San Francisco, CA), ACM, New York, April 1985, pp. 161-167.
- [Fischer, Stevens 87]  
G. Fischer, C. Stevens, *Volunteering Information -- Enhancing the Communication Capabilities of Knowledge-Based Systems*, Proceedings of INTERACT'87, 2nd IFIP Conference on Human-Computer Interaction (Stuttgart, FRG), H.-J. Bullinger, B. Shackel (eds.), North-Holland, Amsterdam, September 1987, pp. 965-971.
- [Hutchins, Hollan, Norman 86]  
E.L. Hutchins, J.D. Hollan, D.A. Norman, *Direct Manipulation Interfaces*, in D.A. Norman, S.W. Draper (eds.), *User Centered System Design. New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, pp. 87-124, Ch. 5.
- [Illich 73]  
I. Illich, *Tools for Conviviality*, Harper and Row, New York, 1973.

- [Johnson, Soloway 84]  
W.L. Johnson, E. Soloway, *PROUST: Knowledge-Based Program Understanding*, Proceedings of the 7th International Conference on Software Engineering (Orlando, FL), IEEE Computer Society, Los Angeles, CA, March 1984, pp. 369-380.
- [Jones, Kapple 84]  
R.J. Jones, W.H. Kapple, *Kitchen Planning Principles - Equipment - Appliances*, Small Homes Council - Building Research Council, University of Illinois, Urbana-Champaign, IL, 1984.
- [Lewis, Norman 86]  
C.H. Lewis, D.A. Norman, *Designing for Error*, in D.A. Norman, S.W. Draper (eds.), *User Centered System Design, New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, Ch. 20.
- [Paradies 73]  
K. Paradies, *The Kitchen Book*, Peter H. Wyden Publisher, New York, NY, 1973.
- [Simon 81]  
H.A. Simon, *The Sciences of the Artificial*, The MIT Press, Cambridge, MA, 1981.
- [Winograd, Flores 86]  
T. Winograd, F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing Corporation, Norwood, NJ, 1986.





Andreas C. Lemke  
Department of Computer Science

---

ECOT 7-7 Engineering Center  
Campus Box 430  
Boulder, Colorado 80309-0430  
(303) 492-1218  
e-mail: andreas@boulder.colorado.edu

# FRAMER

## A Design Environment for Window Frames

Charles Hair and Andreas C. Lemke

### 1. Statement of the Problem

The basic goal of this work has been to construct an improved version of the Frame-Up tool that is available on the Symbolics. That tool permits users to design their own user interfaces by establishing various panes within a frame, where panes can have differing characteristics according to their intended uses.

By "improved" we mean that we want to create a tool that will be more readily usable than Frame-Up by inexperienced users. In order to improve the tool in this sense, we have used direct manipulation, examples, and a critic. An important aspect of this work has involved deciding how to represent design knowledge for inexperienced users.

By trying our implementation ideas out both on ourselves and on other subjects we have refined our ideas about how to more effectively bridge the gap between the user's model and the actual system. To some extent we have also clarified our thinking about the general problem we are trying to solve by identifying three aspects of it. There is first a syntactic understanding problem involved with users' understanding of how things must be done. Then there is a semantic understanding problem which involves users' understanding of why they might want to do various things. Finally, there is the pragmatic problem level in which the user really only wants to know enough syntax and semantics to accomplish a given goal.

In our view, all three of these problem areas need to be addressed in order to have a system that can be used to good effect by new users. Thus, even when a user really only wants to know as little as possible in order to achieve a specific goal, the user will need to know at least some of the syntax and semantics of the system.

### 2. Study of Frame-Up

In addition to noting our personal criticisms, we circulated a questionnaire among the members of the class to try to learn what other people have thought about using Frame-Up.

All subjects acknowledge that Frame-Up is a step forward from using the `dw:define-program-framework`

macro directly. It is faster and requires less syntactic knowledge. Frame-Up however does not reduce the required semantic knowledge. It also does not support a learning process to acquire this knowledge. Hence, subjects complained that they did not understand some of the terminology and the purpose of the different pane types. Some of the commands that are available in Frame-Up are not readily understood. For instance, we only found out a week ago that one of the commands actually makes a couple of example frames available to the user.

Another complaint was that the "Split Panes" command is awkward to use, does not give users the fine control they desired, and makes certain rearrangements impossible without having to start the design from scratch. Panes cannot be directly moved, shaped, or deleted. Frame-Up also does not cover the whole functionality of the macro. It was observed that to do really unique designs one had to directly modify the code involved. Therefore, it is still necessary to understand the macro, which was hard for the subjects.

Another factor that limits the usability of Frame-Up is the fact that once an initial design effort has been saved from Frame-Up, it is not possible to go back to Frame-Up to modify that design. (A copy of the questionnaire is in Appendix A. The results are summarized in Appendix B).

### 3. Rationale

Miller (1978) has studied a similar problem. His goal was to design a "structured planning and debugging environment for elementary programming." He wanted to build a system to facilitate the acquisition of programming skills. Although our goal was explicitly to enable the user to get away with as little learning as possible, there are many common issues. His system (called Spade-0) was designed as a "diligent clerk" and not yet as a tutor. It interacts with the user using a vocabulary of planning and debugging. Although it does not have any apriori specification of the programming task, it builds up a plan representation as instructed by the user. The author notices that the lack of knowledge about the problem severely restricts the support that the system can give.

The Pride system described by Mittal and Araya (1986) attacks the problem of supporting design in well-defined solution spaces. Here, the user gives a partial formal specification of the problem, and the system assists in exploring the large and complex design space. The system has knowledge about constraints that exist in this solution space. Similar to the window frame design problem, Pride supports the exploration of a design space where formal specifications are unavailable. Pride exemplifies how design knowledge can be represented as constraints and heuristics. Pride's constraints are similar to FRAMER's rules.

In order to accomplish our goals, we have tried out a few specific ideas. First, since this problem domain involves building a tool for designing visual objects from visual parts, it appeared to be a natural to use direct manipulation in much the same way as the Pinball Construction Kit. Second, we felt this area was an appropriate one in which to explore the critic paradigm. Third, this area is one in which we believe that it is quite useful to offer users full blown design examples as an aid to their design efforts.

By actually trying our ideas on test subjects we have gained some insight into the kinds of things that really work. Actual observations have led us to see that some things were not as clear as we initially thought they would be to users. These observations have also suggested that examples and critic type capabilities are particularly desirable for novice users.

To summarize, we have learned more about how to better support inexperienced users in their own design activities. We have tried to find techniques to allow nonexperts to make use of the rich set of abstractions offered by the Symbolics lisp machine, as a high-functionality computer system. The system is also usable by experts for less involved tasks. Part of this effort has involved trying to create a tool that is easy to use, and part has involved trying to make the tool intelligent.

#### 4. Technical Approach

This system has been implemented on a Symbolics computer. As part of the implementation we have made use of Frame-Up, flavors, presentation types, and a variety of window panes. These Symbolics tools have had varying impacts on the development of the project.

To a large extent, the Symbolics tools made the project much easier than would be the case in other systems. The existence of presentation types is extremely useful in implementing the direct manipulation aspects of the project. It has also been useful to have different kinds of panes and particularly to have command menus and the mouse documentation line. Frame-Up itself was useful at the beginning of the project when we made our first effort at designing the FRAMER interface.

The direct manipulation aspect is implemented by making a palette of icons available to the user that can be made use of in a separate work area. This approach is similar to that of the Pinball Construction Kit and of CRACK. We have also introduced the use of examples of properly designed frames such that one such example is always shown in a separate pane. These examples can be used directly by the user as a starting point for their own designs, and users are allowed to create and store their own examples. The use of examples is also present in other systems like the Pinball system, and Frame-Up itself, but we have tried to make our examples a more obvious and central part of our system.

FRAMER represents design knowledge in form of rules that the user can invoke. Unlike the approach in the kitchen design project, we have implemented our critic as a completely passive one. Our experience with subjects so far suggests that at least in this kind of context, novice users are fairly willing to use the critic as an aid in finding out what they should do.

#### 5. Description of the Program

The program works by presenting the user with five window panes. One pane consists of the actual work area in which the user constructs an interface. A second pane holds icons, which the user can make use of in building the interface, much like in the Pinball Construction Kit. A third pane permits the user to type in keyboard commands to FRAMER. The fourth basic pane lists the FRAMER commands, allowing those commands to be selected with the mouse. In a fifth pane the user is allowed to access the examples of predesigned frames.

Our overall approach is intended to reduce the amount of knowledge a user must have about either the Symbolics system in general or the domain of designing frames in particular. Thus, through the use of the critic and the examples a user should be able to easily build a professional quality interface without necessarily understanding all of the design decisions made. At the same time, our approach could lend itself to explaining to users why the system is designed as it is, if the users wish to find out.

In using the system, the user can begin either by copying an example into the work area, or the user can build up a frame starting from scratch. Icons can be manipulated with the mouse or through typed commands. The basic manipulations allowed are to move and shape icons within the work area, to copy icons (including whole frames and examples) into the work area, to change an icon label, and to delete an icon from the work area. There are also commands permitting the work area to be cleared and for user created frames to be saved as examples in the catalog. The examples in the catalog pane can be viewed by use of the scroll bar in that pane. Newly stored examples are automatically displayed in the catalog.

The arrangement of panes is supported by the expand-panes-in-frame command and by a snapping facility. The expand command enlarges all panes in a frame so that they occupy all of the frame space. In addition, a mouse-tracking capability causes icons to be automatically snapped together when they are moved close together.

All of the operations on icons can be accomplished with mouse clicks. In addition, all of the operations can at least be initiated with typed commands. In moving and shaping operations the mouse must be used in order to indicate where to move icons and how to shape them.

Our critic operates through the use of a rule base that has rules relating to the correct way of designing a frame. The critic is invoked by using the suggest-improvements command and then indicating which frame advice is being requested for. The generated advice is given in FRAMER's listener pane. When a rule is fired some canned advice is presented to the user about an indicated frame. The advice is presented as a mouse-sensitive advice object. Hence, it is available for further operations. Each rule contains a text describing the rationale for the suggestion. This text can be displayed. A second operation is available on some rules. It allows the user to have the system modify the design according to the suggestion.

The critic has limited design knowledge about title panes, listener panes, and command menus. For title panes, for example, it knows that frames usually have title panes, that they should be at the top of the frame, and that there should be no more than one of them. Upon request, the system knows how to add a title pane if there is none, and how to move it to the top, if it is somewhere else. This knowledge is useful to remind the user of simple rules about frame design that might not be known or forgotten by the user.

## 6. Program Behavior

The program begins by presenting the palette of icons and by showing one of the examples. The screen images in Appendix C illustrate how some of the operations work.

The first image shows the initial configuration. The second image then shows what has happened after the user has placed some icons in the work area. As illustrated, the user is free to place any icon anywhere in the work area, and can obtain multiple instances of the icons. The third image was obtained after the user had deleted the listener icon and had used the command to expand the icons in the frame. The final image illustrates the use of the critic (the advice appears in the listener pane at the lower right), and that the user has copied an example into the work area and modified that frame. In these images the commands that have been executed can be read in the listener pane.

## 7. Evaluation of the Program

It has been enlightening to actually try the system out on other people to see where things are possibly not as good as we thought. These informal experiments gave us direct feedback on how successful our efforts were. In having people try our system we have asked them to do a fairly simple task. We have asked them to set up the interface for an electronic news system in which they would need two panes: one for showing actual messages and one for showing what the possible message categories are. Part of what we were interested in observing was whether subjects would put in a title pane in addition to the two panes directly specified.

One problem that our subjects had was understanding that they had to get a frame into the work area and put icons representing various types of panes into the frame. In a first version of FRAMER, a frame was initially in the work area, but the subject could not recognize the large rectangle with the thick border as a frame. The subject did not know how to proceed and executed the "suggest improvements" command. The displayed suggestion enabled this understanding process and the subject proceeded with few further problems. This observation suggests that users are willing to read instruction when they hit an impasse. For the next version, we added the label "Frame" to the frame icon but presented an empty work area because subjects can get a frame either from the palette or the catalog. This modification notwithstanding, the next subject thought that the working area as a whole was the frame (certainly a reasonable assumption). The visual appearance of our system is probably too poor to project the right system image. Also, it is not clear whether our notion of having frames as objects within the work area is the best approach. Subjects seem to be at least initially somewhat confused about what the frame icons are for.

As could have been expected, subjects with little Symbolics familiarity need to be prompted about things like looking at the mouse documentation line or about how to really manipulate the mouse. We observed interference with differently working mice. One person who was evidently familiar with the Macintosh kept trying to position icons with the Macintosh dragging technique. The slowness of the mouse feedback caused some users to assume that their actions did not have any effect.

Apart from the failure to recognize the frame icon, it is apparent that our iconic representation at least conveys more information than using the Frame-Up method of simply listing the names of the kinds of panes. All of the subjects used a title pane, and all of them succeeded in building the basic frame we asked for.

The experiments have shown that FRAMER goes a long way toward eliminating a lot of the syntactic and semantic problems. We think that it has demonstrated the value of using direct manipulation techniques in appropriate circumstances. There has also been something of a demonstration that direct manipulation can be used to effectively introduce greater functionality in a system while simplifying the means of achieving that functionality.

## 8. Potential Further Developments

One possible further development would be to set up the critic with a graphical capability. Instead of describing the problems of a design verbally, a constructive critic could show improved versions graphically, possibly with an explanation of why the proposed design is to be preferred.

It has been suggested that there should be a facility allowing users to abstractly describe requirements for

the pane design, e.g., by indicating how many panes of which type are required. Also, with this description the catalog of examples could be filtered to show only a reduced number of items. We believe, however, that generating an abstract requirements specification will be at least as difficult as directly constructing a draft version by direct manipulation. Experience has shown that users do not know the requirements in advance, and requirements contain specification errors. The catalog will not contain more than about a dozen predesigned examples. It will, therefore, not be time consuming to peruse the catalog.

We would also like to ultimately go back to our original idea of representing icons at the level of components of panes such as scroll bars and borders, instead of just complete panes. The greater level of detailed design these icons would allow would further reduce any need the user might otherwise feel to go to the code itself to achieve certain goals. Of course, our critic would then need to be updated so that it would have knowledge about how to use such items.

The rule base of the critic component is currently very small and covers only a small fraction of the design knowledge of an expert designer. The passive critic mode worked well in some of our experiments. It needs to be more thoroughly evaluated, however.

Additional further areas of knowledge could include knowledge about frame paradigms other than that of the Symbolics. Examples would be the methods used by MacIntosh and Sun.

FRAMER currently does not actually create or read code. We believe that it would be straightforward to add this functionality, as has been demonstrated in other systems.

We would also want to make a variety of kinds of improved help available. It should be possible to obtain information about the uses of an icon by clicking on the icon and information about a command by clicking on the command. Another desirable feature would be the implementation of a hypertext kind of ability in the critic to allow the user to obtain detailed explanations. There should also be some explanations available about the design features of the examples.

Finally, we have really only touched the surface when it comes to trying our system out on subjects. Not only would it be desirable to have people continually trying out the system for short tasks as further refinements were made, but it would also be a good idea to try to have users try the system out for real life tasks once the system was developed that far.

**Acknowledgements:** We thank Bernie Bernstein, Curt Stevens, Patrick Lynn, Bill Turnbull, Rich Fozzard, and Victor Schoenberg for their valuable comments and for serving as test subjects in our system evaluation study.

## 9. References

- Miller, M.L., (1978). "A structured planning and debugging environment for elementary programming." *Int. J. Man-Machine Studies*, 11, 79-95.
- Mittal, S., & Araya, A., (1986). "A Knowledge-Based Framework for Design." *Proceedings AAAI*, pp. 856-865.

## Appendix I. Questionnaire

### Questionnaire on Frame-up

(Please return to Andreas or Charles)

1. Estimate how many times you have used frame-up: \_\_\_\_\_
2. Estimate how many hours you have spent directly using frame-up: \_\_\_\_\_
3. Estimate how many hours you have spent modifying the code produced by frame-up: \_\_\_\_\_
4. Describe any initial problems you had in using frame-up.
5. Describe anything you still have problems with in frame-up.
6. Describe what you like about frame-up.
7. Describe what you do not like about frame-up.
8. What specific features would you like to see in frame-up?

## Appendix II. Summary of Responses

### Advantages of Frame-Up to the code level:

Less syntactic knowledge required.

Arranging the panes (within what is possible with FRAME-UP) is easier.

Good for rapid prototyping of secondary or new configurations.

Would take much more time to write that code by hand.

Visually oriented.

Current definition can easily be tested to see what it looks like.

Its functionality is apparent and does not require "learning" how to use it.

### Problems:

How to transfer the definition into an editor buffer.

I did not notice that there is a facility to set global framework parameters.

I did not understand some of the terminology.

Did not understand "initial configurations"

Did not understand what the different kinds of panes were (esp. interactor vs. listener).



Did not understand the options: e.g., what does "read single-character command accelerators" mean?

Panes cannot be rearranged.

Margin components cannot be selected.

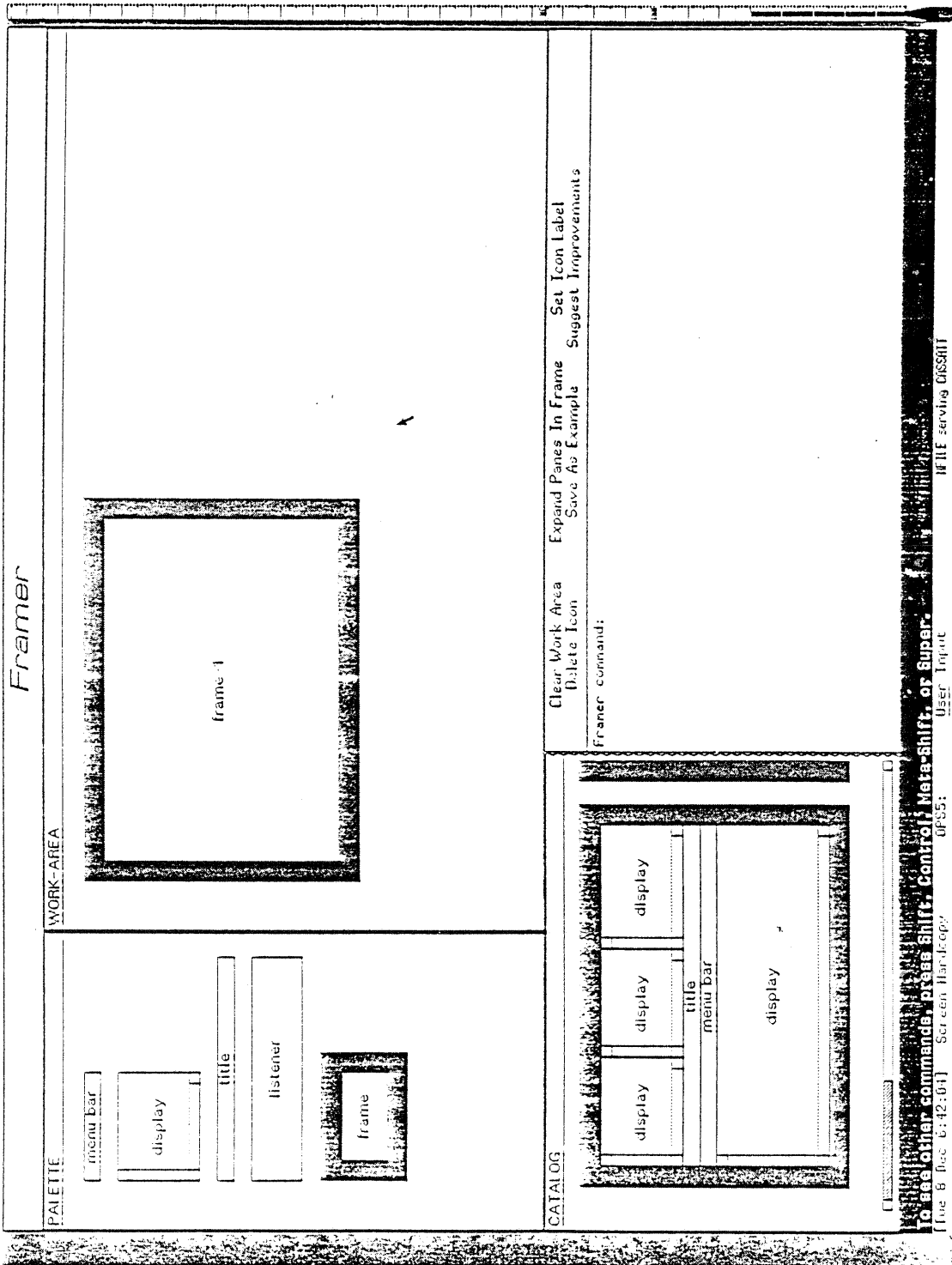
Existing framework definitions cannot be edited with the tool.

I hate the way "split panes" works. Equal size splits.

Code produced hard to read.

# Appendix III. Screen Images

## Initial Configuration



# Icon Manipulation

*Framer*

**PALETTE**

**WORK-AREA**

**CATALOG**

Clear Work Area    Expand Panes In Frame    Set Icon Label    Suggest Improvements  
 Delete Icon    Save As Example

[18:42:09 Process Screen Hardcopy wants to type out  
 Select Screen Hardcopy Background Stream by typing Function-0-S.]  
 Framer command: Copy find Shape Icon "title"  
 Framer command: Copy find Shape Icon "display"  
 Framer command: Copy find Shape Icon "display"  
 Framer command: Copy find Shape Icon "menu bar"  
 Framer command: Copy find Shape Icon "listener"  
 Framer command: █

**To see other commands, press Shift, Control, Meta-Shift, or Super.**  
 [Tue 8 Dec 8:46:08] Screen Hardcopy OPS5: User Input    FILE serving CICSMT

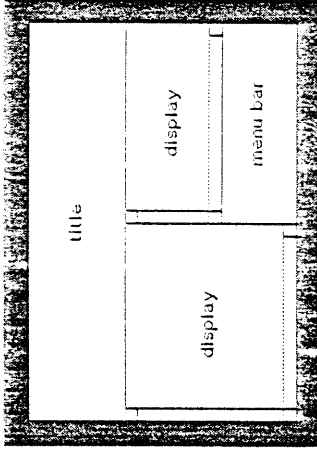
# Expand and Delete

## Framer

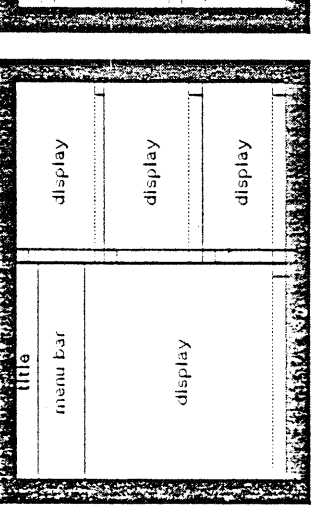
**PALETTE**

- menu bar
- display
- title
- listener
- frame

**WORK-AREA**



**CATALOG**



**Clear Work Area**    **Expand Panes In Frame**    **Set Icon Label**

**Delete Icon**    **Save As Example**    **Suggest Improvements**

Framer command: Copy And Shape Icon "title"  
 Framer command: Copy And Shape Icon "display"  
 Framer command: Copy And Shape Icon "display"  
 Framer command: Copy And Shape Icon "menu bar"  
 Framer command: Copy And Shape Icon "listener"  
 Framer command: Expand Panes In Frame "frame-1"  
 Framer command: Delete Icon "listener"

**To see other commands, press Shift, Control, Meta-Shift, or Super.**  
 [Tue 6 Dec 7:02:48] Screen Hardcopy GP55: User Input

Use Example and Critic

**Framer**

**PALETTE**

**WORK-AREA**

**CATALOG**

**Clear Work Area**   **Expand Panes In Frame**   **Set Icon Label**  
**Delete Icon**   **Save As Example**   **Suggest Improvements**

(18:42:09) Process Screen. Handcopy wants to type out  
Select Screen Handcopy Background Stream by Typing Function-0-S.)  
Framer command: Copy find Shape Icon "title"  
Framer command: Copy find Shape Icon "display"  
Framer command: Copy find Shape Icon "display"  
Framer command: Copy find Shape Icon "menu bar"  
Framer command: Copy find Shape Icon "listener"  
Framer command: Expand Panes In Frame "frame-1"  
Framer command: Delete Icon "listener"  
Framer command: Suggest Improvements "frame-1"  
Rule:NEED-LISTENER-PANE: Hold a listener pane.  
The listener pane allows the manual input of commands.  
Framer command: Renedy FRAMER::NEED-LISTENER-PANE  
Framer command: █

**To see other commands, press Shift, Control, Meta, Shift, or Super.**  
[File 0 (Dec 6:50:10) Screen Handcopy]   **OFFS:**   User Input

## Table of Contents

1. Statement of the Problem	1
2. Study of Frame-Up	1
3. Rationale	2
4. Technical Approach	3
5. Description of the Program	3
6. Program Behavior	4
7. Evaluation of the Program	5
8. Potential Further Developments	5
9. References	6
Appendix I. Questionnaire	7
Appendix II. Summary of Responses	7
Appendix III. Screen Images	9



**NewScope:**

# **Towards a User Modeled Personal Information Retrieval System**

## **Project Report**

**Curt Stevens  
Bernie Bernstein**

**University of Colorado at Boulder**

### **1 Statement of Problem to be Attacked**

The problem of Personal Information Retrieval is as complex as it is unweilding. The questions of how one can best organize and access personal information databases is complicated by the simple fact that each human is very different. Systems designed to help users organize their information must therefore be tremendously flexible in that each user should be able to access information in whatever manner suits him/her best. Moreover, these systems should be able to offer users assistance in finding new information that would be of plausible interest to them. Towards this end we are looking at the problem of user modeled information retrieval.

In order to keep the problem more manageable we have designed and implemented a prototype network news reader. This domain is large enough that we encountered many of the problems associated with very large information bases (eg. finding the interesting information; organizing the interesting information) while still allowing us to deal with a single information base to simplify the process of rapid prototyping.

Our understanding of the problem has changed somewhat from the beginning of the project to now. During the coding effort it has become clear that there is a limited amount of help we can give to users without some sort of sophisticated user modeling capability. This would have the ability to recognize



users' level of expertise, suggest newsgroups that are within the scope of users' interests, as well as doing many of the currently manual chores (eg. defining message filters) automatically. While NewScope does have a primitive user model, in the form of user personalized filters, we are a long ways away from truly intelligent information retrieval. In addition, the questionnaires we handed out have given us much more insight to how and why people read the news database.

In general, the main problem we have addressed is the necessity for wide flexibility in information retrieval systems. We break this flexibility into two distinct categories. The first of these is display flexibility which is concerned with the users' ability to quickly and easily reorganize the display of newsgroups into displays which hold only the information which is desired. The second of these categories is organizational flexibility which is concerned with the users' ability to easily organize and filter the information deemed interesting. The issue of organizational flexibility is addressed by the *virtual newsgroups* component of NewScope. The personalized filters we mentioned above, which create these *virtual newsgroups*, serve as a much more flexible means of determining what are the interesting pieces of information, than is currently available in the UNIX news reading programs (rn, vn, readnews etc.). At the same time, the issue of display flexibility is addressed by the graphical browsing capabilities (ref. Tristan) which display only the portions of the news hierarchy which the user asks for.

## 2 Project Rationale

The reasons that this project is important would be obvious to anyone who is a frequent user of the news system. One of the most influential arguments against making use of the news information base is that it is just too large for anyone to be able to utilize. This was by far the top reason given on our questionnaires for why people don't read lots of newsgroups. This project is looking at how to lessen the impact of the sheer sizes involved. The filtering out of *uninteresting* news groups and messages should be an automatic operation. With the advent of a user modeled filtering system, NewScope could greatly reduce the perceived size of the news system. This is accomplished by only showing users what they are interested in reading.

In addition to the sheer size issues, another complaint about the news system is that it is very difficult to know what news groups are about, and therefore difficult to choose which groups to read. The problem is not that this information doesn't exist, it does, but that people don't know where to find it. A user model can help here by suggesting new groups, or groups which the user has passed up, based on the content of the groups currently being read. It is in these areas that we expect our analysis of the news questionnaire to pay off in the long run by supplying us with insight into the heuristics which users utilize to choose which groups to subscribe.

These are all perfectly valid reasons for exploring this domain, but these functions were really beyond what we have been able to accomplish up until now. However, the parts of the problem which have been attacked so far also constitute an interesting exploration. On the interface side, we have solidified what we feel is a good paradigm for organizing personal information bases like news (eg. large, dynamically

changing information stores). We have combined the browsing mechanisms of systems like Tristan, with a Rabbit like mechanism for retrieval by reformulation (although ours is significantly less sophisticated). These are embodied in the user interface and filtering mechanisms respectively.

On the questionnaire side, we have laid the groundwork for a much more useful user model. One of the biggest problems we have found in formulating a user model is deciding which aspects of user behavior to model. The answers to our questionnaire have gone a long ways towards clarifying this issue by showing us what the actual users of the news system think is important to them. In addition, the questionnaires have given us valuable clues on how we might actually model the aspects of user behavior which we deem important or interesting. (see section 7 for observations from the questionnaire and data gathering efforts)

### 3 Technical Approach

NewScope has been written on the Symbolics machine. It fully utilizes the presentation system and Tristan (a graphical browsing system). The advantages of doing this were numerous. The greatest advantage of using the presentation system was its ability to do interreferential I/O. This is implemented such that the user can easily reference any information displayed by either the computer or the user. All graphical output to the screen can in turn be used, through the mouse, as input to NewScope commands. In addition, the presentation system provides us with a higher level interface to mouse handling routines so that the programming effort was greatly reduced. Finally, the presentation system provides us with the mouse documentation line, which in turn is used as a powerful mechanism to impart information to the user in an asynchronous manner.

There were also advantages to the reuse of Tristan as applied in the NewScope environment. The functionality of Tristan is well suited for the display of hierarchical structures. Clearly the current news system fits perfectly into this paradigm. Again, the existence of Tristan has significantly reduced the programming effort involved in implementing NewScope.

The provision of a naturally occurring hierarchy in the news system was our basis for displaying the information in NewScope as a graphical hierarchy. This does not necessarily mean that a hierarchical decomposition of news is the most natural organizational paradigm. With the addition of more complicated functionality (see section 6) we may find that additional organizational structures are necessary.

Given that we haven't implemented this added functionality, we have fully utilized the hierarchical structure of the current news database. This includes an extension of this structure into the article browser of NewScope. Traditionally, messages are ordered chronologically and successive messages may or may not address related topics. In NewScope, however, a hierarchy of messages is created and responses to previously posted messages are displayed as children of the original posting. Part of our interest in NewScope is based on the ability to sift out only the interesting information. Since this implies that there will be many related messages, this hierarchical organization of messages helps the user to visually understand these relations and quickly access them.

There are other systems which have attempted to solve similar issues to those which we are exploring here. One of these systems which seems to hold much promise is the **The Information Lens**, written by Thomas Malone and his research group at MIT. There are two major differences, however, between what we are doing and what they were doing in that system. First, the database they were working with was personal mail systems. While many of the characteristics of mail are also present in the news system, it is the personal nature of mail which sets our study apart from theirs. Except for the LENS mail alias (discussed below), Malone's system deals with sifting through information which is known to be meant for the person reading it. This is exactly the opposite situation than we have in news, where almost no messages are directed to a particular person.

The second major difference between these systems involves the community database (somewhat resembling local newsgroups) which is posted to by including the alias **LENS** on the distribution list of a message when sent. In the news system there is a predetermined hierarchy defined into which users must categorize their postings. In the LENS system, even after users go through the process of categorizing their message in order to get a template, all public messages are sent to a single mailbox and string searching techniques are used to filter the information to users who have requested it with user defined rule sets.

#### **4 Let's See it in Action: Scenario**

We will assume for the purpose of this section that the user is interested in reading news about the Macintosh II, especially news about using color graphics on this machine. In addition, for the sake of simplicity, we will assume that the user has already created a virtual newsgroup (`comp.sys.mac.mac2`) to sift out this information from the `comp.sys.mac` newsgroup (the virtual newsgroup consists of messages contained in a parent newsgroup which have passed through a user defined filter in order to create a new newsgroup which really doesn't exist in the news heirarchy, but which contains user specified types of messages).

The user begins the session by selecting the **Display Newsgroup** command from the main menu and entering `comp.sys.mac` as the desired newsgroup. He may have done this to be sure not to miss interesting information which is generally about the Macintosh. Once this newsgroup has been displayed in the browser, he chooses to expand its subgroups (see screen dumps). Being especially interested in color graphics, he immediately moves down the hierarchy to the `comp.sys.mac.mac2` newsgroup. At this point the user enters the article browser by requesting to see the messages within this virtual newsgroup. This is all done with two or three clicks of the mouse.

Once in the article browser, the user sees a hierarchical organization of related message topics. Fortunately, it seems that there is information about color on the Macintosh II and he chooses to display the articles related to the topic *Mac II color icons*. Responses to the original posting of this message are conveniently attached to it by means of the graphical display. An arrow links the original message with its responses. This is how the hierarchical organization of the news database in graphical format helps the

user to more quickly and efficiently access related information. In traditional news systems (m...) the user would have to browse over numerous unrelated messages, or utilize a cryptic keystroke command, in order to read only the desired messages.

In addition to being able to quickly traverse the hierarchy, the graphical display of the subtopics of comp.sys.mac.mac2 on a single screen allows the user to quickly find subjects which are uninteresting and therefore constitute good candidates for the filtering mechanism. For example, this user no longer wishes to read articles about buying a Macintosh II. He therefore invokes the filter mechanism to prevent the future display of any messages relating to this topic (this is essentially removing topics from this virtual newsgroup and returning them to the parent newsgroup). Now the user looks at his watch and realizes it is time for class and exits NewScope.

## 5 Evaluation of program

The NewScope system in its current incarnation executes all of its functionality quite well. Users can freely browse newsgroups, browse articles, and create virtual newsgroups (which functions to both include interesting information and exclude uninteresting information). The major drawbacks of the NewScope system mainly relate to the functionality which is clearly needed but not yet implemented. This includes virtual newsgroups which inherit from different branches of the newsgroup hierarchy as well as the more sophisticated user modeling capabilities we mentioned earlier. Also, like all research efforts on Symbolics machines, the NewScope system would benefit greatly from improved system performance. Waiting 15 seconds for a menu is totally unacceptable in a real world application.

While NewScope does exhibit these shortcomings, it still serves to help clarify some important theoretical issues. As we mentioned in section 1, one of the overriding concerns with the development of large dynamic databases like news is the problem of organizing vast amounts of information. NewScope, with its filtering mechanism, can greatly reduce the perceived size of the information space. In addition, the higher level of organizational stability created by graphically representing the newsgroup hierarchy serves to make browsing this information a much more efficient and enjoyable process. This is clearly important in a system like news where much of the activity is meant to be enjoyed as a break from normal work. The UCSD book (User Centered System Design) has stated that these types of systems will usually fail if they are not truly easy and enjoyable to use. Any efforts we as system designers can make to reduce the cognitive load on users of these systems will benefit both user and designer in the long-run.

## 6 Future Shock

There are numerous enhancements we would make to NewScope if we had more time to work on it. Some of these have already been mentioned throughout this report. Perhaps the most interesting of these would be the addition of a hypertext component to the news system. While this would undoubtedly necessitate additional data-structures, it would also, most probably, necessitate the addition of some sort of rule-base to the system as well. Since the senders of a message have no idea what other messages

will be in the database when their message is posted, they cannot set up links to related items. Therefore, there will have to be a component of NewScope which can effectively make these hypertext links in a dynamic fashion as new news arrives. In addition, this component will have to interface with the filtering mechanism in order to remove links to *uninteresting* information.

Hypertext, however, is not the only potential enhancement which might require a knowledge-base to implement. We would also like to see the addition of an intelligent filtering mechanism capable of both creating virtual newsgroups based on a user model (currently a manual process performed by the user), and also determining the appropriate place for messages to be placed when the user wants to save them. These components, of course, raise many new issues which will make for interesting investigations.

There are also less elaborate mechanisms or features which we would like to implement in NewScope which would not require a knowledge-base to be effective. For one thing, the filtering mechanism should be able to create virtual newsgroups which inherit their messages from more than one parent newsgroup. Since the news hierarchy is littered with related newsgroups that are located in totally different parts of the hierarchy (this is evidenced by the necessity for a cross-posting component in the current news system), users should be able to flexibly organize this related information into single virtual newsgroups.

These less elaborate features also include a DYK-like mechanism for dynamically presenting interesting messages to users as a break from reading the normally subscribed to newsgroups. This might be a very effective way to get users familiar with newsgroups which are not currently subscribed to.

## 7 Appendix A: News Analysis Details

This section will present some of the information we have gathered about the news system during our efforts so far. Unfortunately, since it takes time to gather enough statistics to recognize trends, we have not had the opportunity to analyze as much of the raw data as we would liked to have analyzed. The results of this problem might tend to make this section of the report seem haphazard and ill-conceived, but this is really not the case at all.

### 7.1 Some Numbers: A Quantitative Analysis

The numeric data we have gathered this semester come from four sources. The first of these was the questionnaire which has already been mentioned. While most of the data garnered from these questions was qualitative, not quantitative, there were a couple of questions which looked for numeric responses. The second and third sources were both analyses which are posted periodically to the very network which we are currently studying. Both of these studies post their results to the newsgroup **news.lists**, and have done so for some time (note: past data was not archived so was unavailable to us here at CU). The final source of our data was produced by a program running here (on *boulder.colorado.edu*) and indicates local news reading trends which were verified by the questionnaire.

Much of the numerical data which we have gathered follows starting on the next page:

Over a two week period...

19948 articles, totalling  
33.786759 Mbytes, were submitted from  
2291 different Usenet sites, by  
6369 different users, to  
376 different newsgroups.

296 articles, totalling  
0.376505 Mbytes, were submitted from other sites (e.g. the ARPANET).

Category	Count	Mbytes	Percent
comp	6991	14.319736	42%
rec	7583	9.477052	28%
soc	2904	5.306719	15%
talk	2145	3.393961	10%
misc	2004	2.871488	8%
sci	722	1.114028	3%
alt	545	0.830872	2%
news	337	0.740079	2%
unix-pc	48	0.159824	0%
bionet	19	0.052519	0%

And over a one month period including the same two weeks as above...

	This Sample	Estimated for entire net
Sites:	646	8400
Fraction reporting:	7.69%	100%
Users with accounts:	82556	1073000
Netreaders:	18796	244000
Average readers per site:		29
Percent of users who are netreaders:		22.77%
Average traffic per day (megabytes):		2.660
Average traffic per day (messages):		1317
Traffic measurement interval: last		21 days
Readership measurement interval: last		75 days
Sites used to measure propagation:		500

If we look carefully at the previous data, we can see just how incredible the volume of news which is proliferated across the world in a typical two week period really is.

2 KB (i.e. 2000 characters) per page

1 MB ~ 500 pages

33 MB = 16500 pages = approx. 4x the Symbolics Documentation

Normal Readers: 300 - 400 words per minute (of non-technical information)

Skimming / Speed Reading: 500 words per minute  
(approx. 1 page per minute)

16500 pages = approx. 280 hours = approx. 11.5 days  
(over a 2 week period)

As we can see from these numbers, it would be next to impossible for the average reader to consume all of the information which comes across the network. This presents us with two of the biggest problems to tackle; namely, how do we find where the information which we find useful is located (eg. what newsgroup) and once we find it, how do we filter only the messages we wish to read into our display. Fortunately, the problem seems to be lessened, or at least we seem to have hope for solving this type of problem, when we examine the qualitative data that was also gathered this semester.

## 7.2 Some Interesting Observations: A Qualitative Analysis

We have gathered more qualitative information than could possibly be discussed in this paper (and remain within the page limit) so we will only discuss a couple of the most interesting observations here. As the project continues, the information which we are skipping now will be expanded and used to create a user model which utilizes the trends we have noticed. Also, more of these observations will be discussed at the project presentations.

Two particularly interesting observations were made during our data analysis. The first of these was taken from the user questionnaires and indicates that the average user subscribes to only 8 or 9 newsgroups (see Figure 1). The reasons for this usually fall into the category of *there is just too much information to read, so I only subscribe to a couple of groups*. There are two things to notice about this: (1) that these users are totally justifying the exploration of this domain, and (2) that unless the volume of news decreases substantially, the tastes of individual users will be easier to determine since a low number of subscribed to newsgroups for this reason implies that user interests (what the user wants to read, not what the user is interested in) may be relatively narrow and therefore easy to determine with a user model.

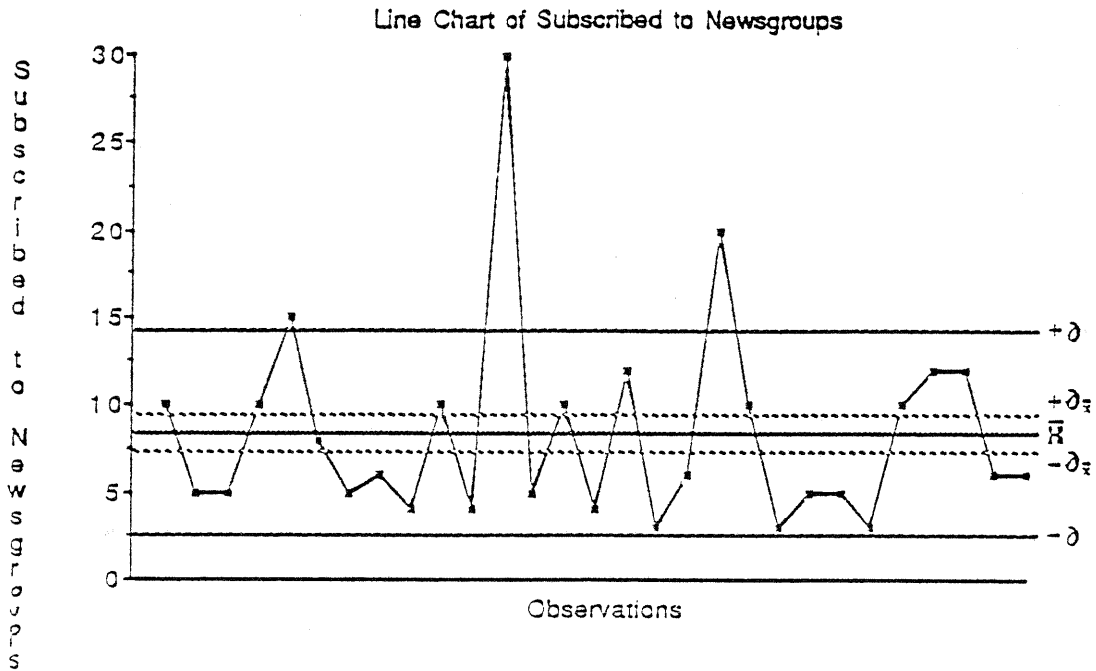
Also, we were able to examine which groups were in the top ten/twenty based upon both popularity and sheer volume of messages. When examining the top 20 newsgroups based on popularity, we found that the most popular newsgroups remained completely constant over a two month period. The exact same 20 newsgroups were in the top 20 by popularity in both of these months. While we don't expect this to be the case every month, it indicates definite trend in the popularity of newsgroups which can definitely be exploited in the implementation of a user model. Even better than that, is the fact that only 5 of these 20 newsgroups show up in the list of top 10 newsgroups by volume. This indicates that the topics of most interest are actually the smaller newsgroups, and that any intelligent filtering mechanisms which are designed in the future will be able to more accurately categorize these newsgroups.

There, of course, is much more information which we gathered, and have not yet had the opportunity to examine in depth. We have no doubt that many more interesting observations wait for us just under the surface of these pages and pages of data.



## Table of Contents

1 Statement of Problem to be Attacked	0
2 Project Rationale	1
3 Technical Approach	2
4 Let's See it in Action: Scenario	3
5 Evaluation of program	4
6 Future Shock	4
7 Appendix A: News Analysis Details	5
7.1 Some Numbers: A Quantitative Analysis	5
7.2 Some Interesting Observations: A Qualitative Analysis	7



Subscribed to Newsgroups					
Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
8.414	5.785	1.074	33.466	68.755	29
Minimum:	Maximum:	Range:	Sum:	Sum Squared:	# Missing:
3	30	27	244	2990	0

FIGURE 1

NEWSCOPE:09/17/87

Newsgroup Browser

University of Colorado, Boulder

Current Message

System Messages

Clear

Display Newsgroup

Redisplay

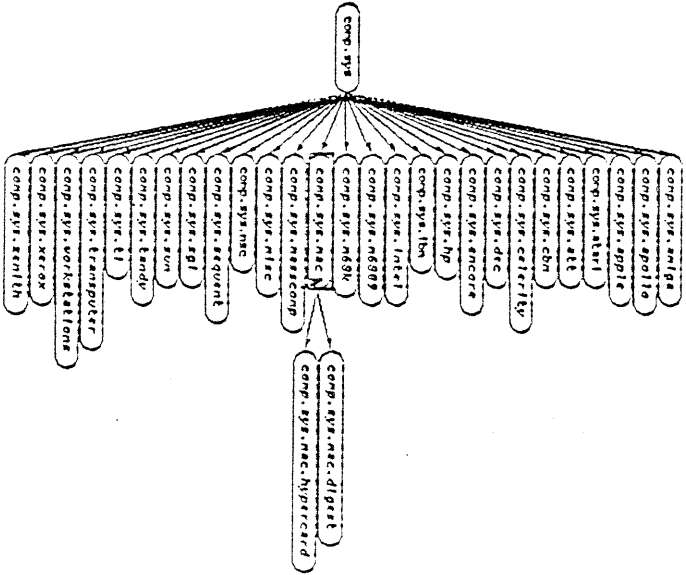
Newscope command: █

NEWSCOPE:09/17/87

University of Colorado, Boulder

NewsGroup Browser

Current Message



System Messages

Clear

Display NewsGroup Redisplay

NewsScope command: █

Home-It: Home menu.  
To see other commands, press Shift, Meta-Shift, or Super.  
Wed 2 Dec 8:17:33j    bernard    CL-USER:    User Input

Article Browser

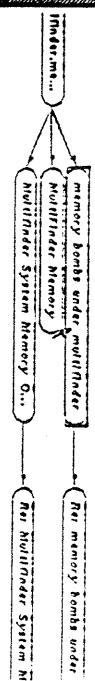
Current Message

From boilder@colorado.edu  
 To: billj@colorado.edu  
 Subject: Multitinder, sys.nac  
 Message-ID: 2272@colorado.edu  
 Date: Fri, 17 Dec 87 16:23:00 GMT  
 Organization: U. Texas, Austin, Austin, TX  
 Lines: 16

I've been playing with multitinder for a few days now, and it's very nice. But the way it handles desk accessories is the pits. All desk accessories that look at byte/char counts to alter what the underlying program actually gets are ambiguous under multitinder. This means that ones such as SmartNotes (which puts proper curly quotes into text--useful with Word) and RedLightning (real-time spell-checker) don't work any more.

Apple, I know you know this will go away given enough time, but this is the wrong way to do things. PLEASE give us back our desk accessories the way they used to be.

Thank you for listening.  
 Bill Jefferys



System Messages  
 Showing list of messages  
 Reading list of 11 total items  
 Reading list of 11 color items

- Jan 07:11, Wed 10:18  
 0101010101 comp.sys.mac.multitinder 1 48 articles  
 reading multitinder.doc
- Jan 02:06, Wed 10:18  
 0101010101 comp.sys.mac.multitinder.memory 1 8 articles  
 reading sys.nac.multitinder.memory

Clear Articles    Newsgroup Browser    Redisplay Articles

Newscope commands: Browse Articles comp.sys.mac.multitinder.memory  
 Newscope commands: █

Home-It: Hide menu.  
 To see other commands, press Shift, Meta-Shift, or Super.  
 Wed 2 Dec 9:01:15J    bernard    CL-USER:    User Input

NEWSCOPE:09/11/7/87

University of Colorado, Boulder

NewsGroup Browser

comp.sys.mac.nsc2  
comp.sys.mac.nsc.nslfinder  
comp.sys.mac.digest  
comp.sys.mac.hypercard

comp.sys.mac.nslfinder-memory

Current Message

Path Boulderheadpa@schu.edu:news:comp.sys.mac.nslfinder-memory  
From bill@sketchy.ucsf (William H. Jefferys)  
Newsgroup: comp.sys.mac  
Subject: nslfinder does  
Keywords: nslfinder, desk accessories  
Message-ID: <22726@sketchy.UCSF>  
Date: 24 Nov 87 16:23:08 GMT  
Organization: U. Texas, Helmsley, Austin, TX  
Lines: 16

I've been playing with nslfinder for a few days now, and it's very nice. But the way it handles desk accessories is the pits. All desk accessories that look at keyboard events to alter what the underlying program actually gets are affected under nslfinder. This means that ms such as pointboxes (which puts proper curly quotes into text--useful with word) and highlighting (real-time spell-checker) don't work any more.

Apple, I know you hope this will go away given enough time, but this is the wrong way to do things. Please give us back our desk accessories the way they used to be.

Thank you for listening.

Bill Jefferys

System Messages

Nov 21 16:13:19 nslfinder 1 29 articles  
reading nslfinder  
reading nslfinder  
Nov 21 16:13:19 nslfinder 1 19 articles  
reading nslfinder done  
Nov 21 16:13:19 nslfinder 1 19 articles  
reading nslfinder done

Clear

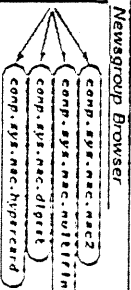
Display NewsGroup Redisplay

NewsScope command: █

House-It: Home menu.  
To see other commands, press Shift, Meta-Shift, or Super.  
Wed 2 Dec 8:58:18J Bernard CL-USER: User Input

# NEWSCOPE:09//AZ/RZ

Operation on comp.sys.nac.multifinder:



Bury the node  
 Bury the superhierarchy of the node  
 Create Virtual NewsGroup  
 Link the node  
 Have the node

University of Colorado, Boulder

out Message

boulder!boulder!colorado!general!dot!early!dot!dot!bill  
 bill!boulder!uwr (William H. Jefferys)  
 color!comp.sys.nac  
 cli!multifinder!bbs  
 ider!multifinder!desk!accessories  
 se!-!Dr! 22178!color!uwr!  
 ORG) 24 Nov 87 16:23:06 GMT  
 Organization: U. Texas, Astronomy, Austin, TX  
 Lines: 16

I've been playing with multifinder for a few days now, and it's very nice. Not the way it handles desk accessories is the pits. All desk accessories that look at keyboard events to alter what the scrolling program actually does are buggy under multifinder. This means that my such as Spreadsheet (which puts proper curly quotes into text-courtesy with bold) and highlighting (real-time spell-checker) don't work any more.

And, I know you hope Dns will go away given enough time, but this is the wrong way to do things. PLEASE give us back our desk accessories the way they used to be.

Thank you for listening.

Bill Jefferys

### System Messages

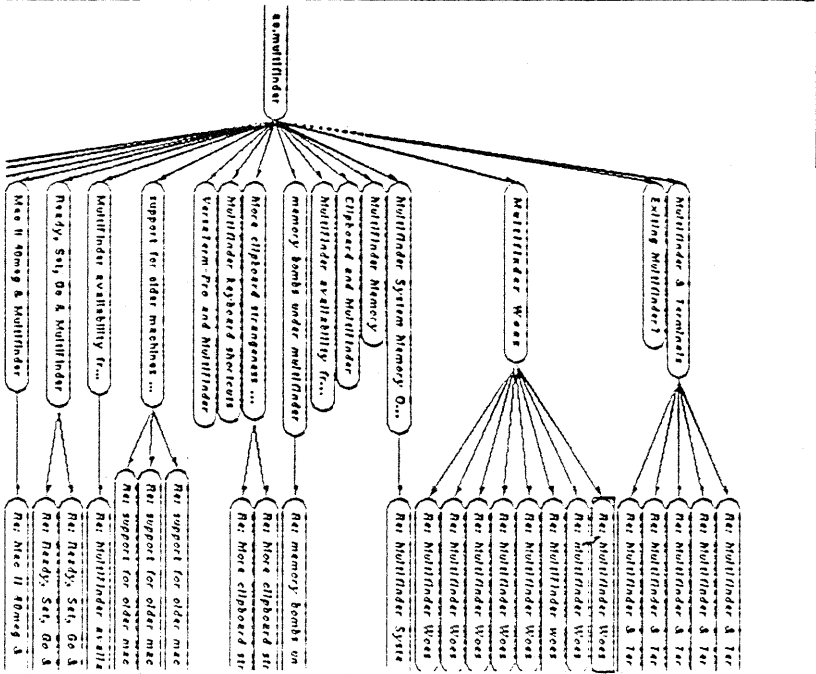
Jan 3178, 11pm 18113  
 0101ay001 comp.sys.nac.naz2 1 28 articles  
 reading Mac II (01-11-87)  
 reading Mac II color found  
 Jan 3177, 11pm 18118  
 0101ay001 comp.sys.nac.multifinder 1 48 articles  
 reading multifinder news

Display NewsGroup Redisplay

Clear  
 NewsGroup commands: NewsGroup Browser  
 NewsGroup commands: Make Virtual Group comp.sys.nac.multifinder  
 NewsGroup commands: █

Home-L, -H, -R: Create Virtual NewsGroup.  
 To see other commands, press Shift, Meta-Shift, or Super.  
 Wed 2 Dec 8:55:58J bernard User Input

Article Browser



Current Message

Path: boulder!healga!edhd-aj!rod!gwal!fotol!cent!pro!ast!ro!bill!l  
 From: bill@ast.ro.UTK (William H. Jefferys)  
 Newsgroups: comp.sys.mac  
 Subject: Multifinder woes  
 Keywords: multifinder desk accessories  
 Message-ID: <227290a@ast.ro.UTK>  
 Date: 24 Nov 87 16:23:00 GMT  
 Organization: U. Texas, Astronomy, Austin, TX  
 Lines: 16

I've been playing with multifinder for a few days now, and it's very nice. Not the way it handles desk accessories is the pits. All desk accessories that look at keyboard events to alter what the underlying program actually gets are annoying to me. I prefer early queries into look-... useful with find and highlighting (real-time spell-checker) don't work any more.

Yeah, I know you hope this will go away given enough time, but this is the wrong way to do things. Please give us back our desk accessories the way they used to be.

Thank you for listening.  
 Bill Jefferys

System Messages

Nov 23 19:25, 11pm 18132  
 018132: [M] CDP: [M] PAC: [M] 1 39 articles  
 018131: [M] CDP: [M] PAC: [M] 1 39 articles  
 018130: [M] CDP: [M] PAC: [M] 1 39 articles  
 Nov 23 19:25, 11pm 18130  
 018130: [M] CDP: [M] PAC: [M] 1 48 articles  
 018129: [M] CDP: [M] PAC: [M] 1 48 articles

Clear Articles

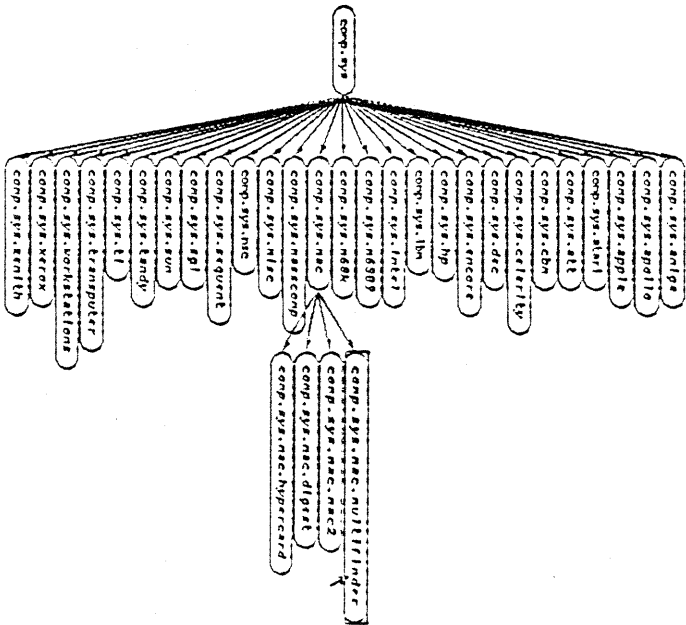
NewsGroup Browser Redisplay Articles

Newscope command: Redisplay Articles  
 Newscope command: Redisplay Articles  
 Newscope command: Redisplay Articles

Hours-It: Home menu.  
 To see other commands, press Shift, Meta-Shift, or Super.  
 Wed 2 Dec 8:52:15J Bernhard User Input



Newsgroup Browser



Current Message

Path: boulder:hae:hae:61uc-sally:ut-comp:ut-rwd:1ec  
 From: jrc@ut-comp:ut-rwd:1ec (Chris Cooley)  
 Organization: comp.sys.nec  
 Subject: Re: Hae II Color Icons?  
 Summary: Yes,...

Message-ID: <23964-enc.UCR>  
 Date: 2 Dec 87 06:13:22 GMT  
 References: <812012072.00230594decv1.dec.com>  
 Organization: the University of Texas at Austin, Austin, Texas  
 Lines: 73  
 Posted: Thu Dec 2 06:13:22 1987

In article <812012072.00230594decv1.dec.com>, harrow@agps1a.dec.com (Lair Harrow, WSE  
 8681-24202, D1H-23-5129) writes:  
 1) do you use only color icons that I've seen on the Hae II in the  
 2) do you use the "arrow" window to machine's dialog box at system  
 3) startup time.  
 4) It's my impression that any icon on the desktop (or used within a  
 5) program) can also be a color icon, but has anyone come up with an  
 6) editor that will create/del/patch color icons?

I was treated to a colorful surprise when I installed the new system (4.7)  
 onto the Hae 2. It offered a menu to the right of the "Special" menu  
 called "Color...". Selecting an icon and then selecting a color from this  
 new change that icon to the color selected.  
 It allows only one color per icon. Maybe even there'll be ways to do  
 all 256 colors in an icon (well, as many as there are pixels in an icon...)

--Chris

System Messages

Thu 8712 06:13:22  
 displaying CPT-115 PGM:03 : 88 articles  
 reading Hae II Color Icons?  
 reading Hae II Color Icons?  
 from 8737, Mon 12 18:58

Clear

Display Newsgroup Redisplay

Newscope command: Read Article Re: Hae II Color Icons?  
 Newscope command: Newsgroup Browser  
 Newscope command: Make Virtual Group comp.sys.nec  
 Newscope command: █

Mouse-II: Node menu.

To see other commands, press Shift, Meta-Shift, or Super.

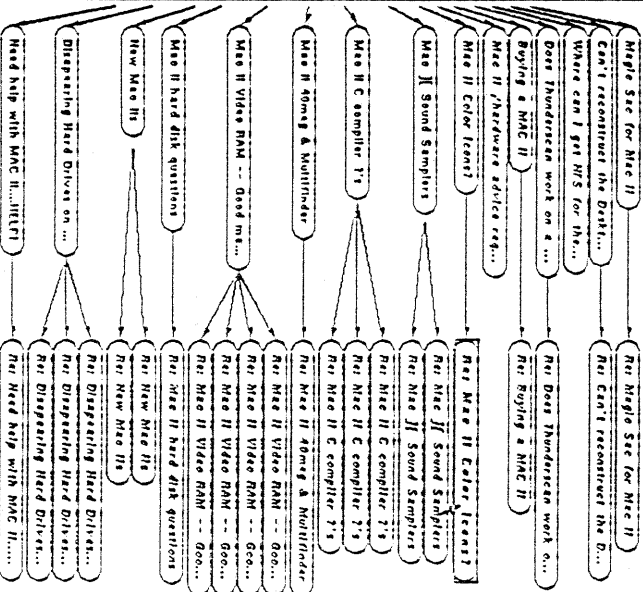
Wed 2 Dec 87:13:31

Bernard

CL-USER:

User Input

Article Browser



Current Message

Path: boulder-hack-hack-hack-!rft-actlypt-!rput-!m1j/c  
 From: Jacob-!em-!MCR (Chris Cooley)  
 Newsgroup: comp.sys.mac  
 Subject: Re: Mae II Color Icons?  
 Summary: Yes...  
 Header: 1b1 473904-em-!MCP  
 Date: 2 Dec 87 06:13:22 GMT  
 References: 4872012022,00335594decur1,drc-com  
 Organization: The University of Texas at Austin, Austin, Texas  
 Lines: 23  
 Posted: Wed Dec 2 06:13:22 1987

In article <812012022,00335594decur1,drc-com>, harrow@utels.dcc.com (Jeff Harrow, W5SE B601-22672-BH-233-5129) writes:  
 > So far, the only color icon that I've seen on the Mac II is the  
 > icon for the default 'Welcome' to Macintosh dialog box at system  
 > startup time.

> It's my impression that my icon on the desktop (or used within a  
 > program) can also be color icon, but my system crew is up with an  
 > editor that will create/edit such color icons!

I've tried to a colorful surprise when I installed the new system (4.2) onto the Mac II. It offered a menu to the right of the "Special menu" called "Color.". Selecting an icon and then "reflecting" a color from this menu changes that icon to the color selected.

It allows only one color per icon. Maybe soon there'll be ways to do all 256 colors in an icon (well, as many as there are pixels in an icon...)

--Chris

System Messages

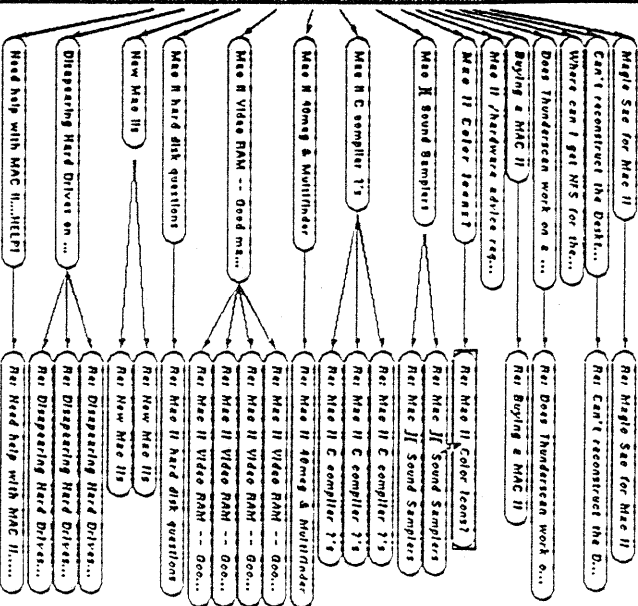
Jan 11/87, Fri 11/12  
 01:04 PM Re: [13] RE: MAC II'S articles  
 Reading Re: Mac II Color Icons?

Clear Articles     Newsgroup Browser     Redisplay Articles

Newscommand: Read Article Mac II Color Icons?  
 Newscommand: Read Article Re: Mac II Color Icons?  
 Newscommand: █

News-n: Node manu.  
 To see other commands, press SHIFT, Meta-SHIFT, or Super-User Input  
 Wed 2 Dec 87:11:27J     Bernard     CL-USER:     User Input

Article Browser



Current Messages

path: boulder-thunderman@ibm1.researcher.com || btkcomp@u11py.enr1ddec.u1bop1a.dec.com  
 harrow  
 From: harrow@hamp@u1bop1a.dec.com (Chrf Harrow, HESG BOU1-2-F07 BU11-233-5120)  
 Message: comp.sys.mac  
 Subject: Mac II Color Icon?  
 Harrow - In: 0872012922.002355@decu1bop1a.dec.com  
 Date: 1 Dec 87 20:22:07 CDT  
 Organization: Digital Equipment Corporation  
 Lines: 23

So far, the only color Icon that I've seen on the Mac II is the one on the default "Welcome to Macintosh" dialog box at system startup time.

It's my impression that my Icon on the desktop (or used within a program) can also be a color Icon, but has anyone come up with an editor that will create/edit such color Icons?

Thanks.  
 Chrf Harrow

Mail address:  
 Harrow  
 IBM/DEC/BOU1-2-F07/DEC/BU11-233-5120 or  
 dirsvr1@vopds.dec.com || harrow  
 or  
 btkcomp@u11py.enr1ddec.u1bop1a.dec.com  
 6600311-57308  
 5117-8-Ed/18  
 Ball Spring Bldg-2-682  
 85 Sunning Road  
 Boulder, IN 81119

System Messages

Mon 11/23 19:41:53  
 Displaying current messages in an article  
 reading Mac II Color Icon?

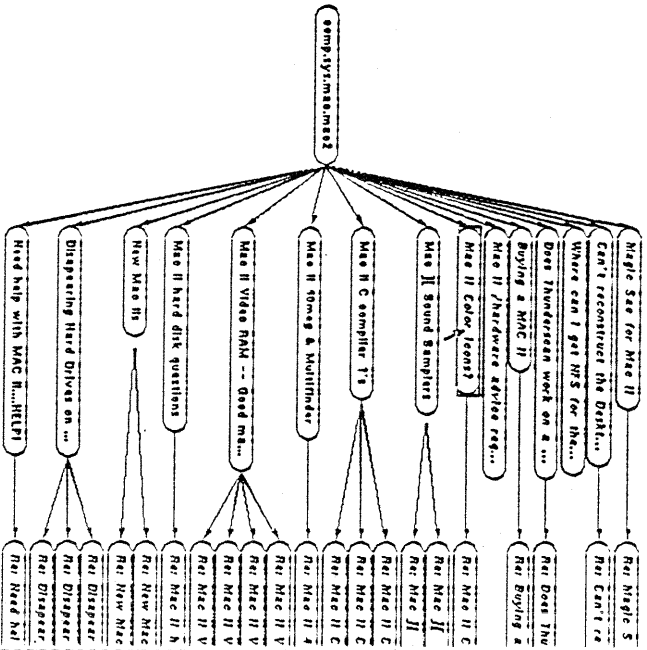
Clear Articles Newsgroup Browser Redisplay Articles

Newscope command: Read Article Mac II Color Icons?  
 Newscope command:  
 Newscope command:

News-H: Mode menu.  
 To see other commands, press Shift, Meta-Shift, or Super.  
 [Wed 7 Dec 8:39:31] Bernard CI-USER: User Input

Article Browser

Current Message



System Messages  
 1001 1125, WPM 1113  
 0101191001 comp.sys.mac.mae: 89 articles

Clear Articles  
 NewsGroup Browser Redisplay Articles  
 NewsScope command: █

House-II: Node menu.  
 To see other commands, press Shift, Meta-Shift, or Super.  
 Used 2 Dec 8:35:23J bernard CL-USER: User Input



## Experiences with the Genera User Interface Facilities

Curt Stevens and Andreas C. Lemke

March 1, 1988

**Introduction** -- In the systems that we have built recently, the presentation substrate has been heavily used. FRAMER and NEWSCOPE, as described in the included papers, each use the presentation substrate to varying degrees. We used Frame-Up to create initial program framework definitions for both systems. This is not however, the only context in which features new to Genera 7 have been utilized. In Framer, for instance, every object on the screen is described by a presentation type. Framer is based on the direct manipulation of graphical objects. Presentation types make this easier by allowing us to create higher level descriptions of our display objects, in the sense that the facilities for activating presentations (e.g., *present/accept*) are already provided. Also, in the direct manipulation environments which we are developing, it is extremely important to offer users multiple modes for input. For example, the Genera environment (if programmed properly) allows users to utilize mouse-clicks, menu-selections, and traditional command language syntax all to achieve equal results (except perhaps for the movement of objects on the screen). These modes do not only aid the users of these systems. One of the main beneficiaries of this modality is the programmer himself. The degree to which these modes have been integrated vastly increases their utility. Specifically, a command can be partially specified by typing from the keyboard, and then completed by indicating its arguments through mouse clicks. This type of interaction basically *comes for free* in Genera 7 and is utilized by any prudent programmer.

**Multiple Representations** -- The Genera environment allows programmers to easily create systems that exhibit concepts and theories which have heretofore been very difficult to implement. One of these powerful concepts is that of multiple representations of a single object. The presentation system makes it easy to have two displays of the same underlying object (e.g., a flavor instance). Since a presentation is separate from the object it represents, we can have graphical and textual representations of a single object. It is important that when a user refers to either one of these representations (e.g., the one which most closely matches the user's model of the system), that we can offer the same functionality to the user in either case. This is possible because the separation mentioned above means that the actual data object being represented in both cases is identical. Every aspect of the object is the same, except the external representation of that object.

**General Shortcomings** -- This does not mean, however, that we feel Genera 7 meets all our requirements equally. One of the biggest deficiencies with the presentation system is the lack of operations designed to act upon presentations, and not the object being represented by the presentation. The example that has caused us the most problems so far is the inability to move a presentation. While we have been able to *kludge* our way around this by examining disassembled code and writing our own functions to move presentations around the screen, there is a definite need

for a metaphor that includes presentation operations for graphically oriented applications. Included in these might be functions which allow the incremental respecification of a currently displayed presentation. In complex presentations which include many inferior presentations, the modification of any part of the presentation causes the removal of the old presentation and the actual display of an entirely new presentation in its place. This causes an awkward visual effect, which often leads users to believe that much more is going on than is actually the case. This leads to much confusion and misconceptions in novice users of any system.

**Specific Shortcomings** -- There are also some specific problems that we have found particularly irritating. The first of these is related to the tracking-mouse function. It is often the case where one wants to track the movement of the mouse between more than one dynamic window. An example of this can be seen in the Framer system. The action required by the user in order to create a new element in the program framework is the selection of a component from the *palette* pane and the subsequent *dragging* of this item to the *work-area* pane. The problem is that tracking-mouse can only be attached to a single dynamic window. Since the whole screen is not itself a dynamic window, we are put in a position where we have to *warp* the mouse (i.e., place the mouse cursor without physical movement of the actual mouse) to the destination pane. This trick worked only because the destination happened to be uniquely defined for all operations in Framer. We view this as a serious problem since moving the mouse cursor under system control generally causes users to lose track of it.

Our second specific problem is that of speed in the generation of menus. While slowness is expected on the Symbolics machines by anyone who knows the complexity of the functions provided, the generation of menus is the one place where the lack of speed is most harshly felt by users. Novice users often think they have done something wrong because the expected menu has not been generated. They click so many times to try and get a response that when the menu finally does appear, the user may have caused unknown effects through queued mouse actions. This is tremendously confusing and the solution is to get *used to waiting for menus*. We feel that this situation must improve especially for the delivery of applications to production environments.

**Learning the Presentation System** -- Another, less specific problem, is related to our ability to effectively use the Symbolics environment for rapid prototyping. The presentation system, as is the case with any system of this complexity and innovative functionality, is very confusing at first. There is a tremendous learning curve which is faced by all new users of the system. We are concerned that there is no support (other than the published reference documentation) that helps to alleviate this problem. We see the future inclusion of some example-based system as necessary to the success of the Genera environment. What we mean here is a system which addresses the following situation. In the documentation which Symbolics provided, there are many definitions of what a particular function does. When each of these functions was written by someone at Symbolics, the author had a particular problem or set of problems which the function in question was designed to help solve. This type of information is invaluable to the intuitive use of these functions but is exactly the information which is not provided in documentation. What is seriously needed is a comprehensive set of examples which use this extra information in context. The lack of examples can double the time it takes to implement a small prototype interface. While operating system source code has traditionally played the example role, we are not implying that Symbolics should include system source code with every distribution (in fact we realize that this is really implausible for any business venture). We do, however, believe that examples could be made available which are much more useful than the current ones (e.g., employee example). This can be accomplished through the introduction of many more examples which are around the same level of complexity as the existing ones, or by the inclusion of a large sample system which includes much more of the system functionality which is available through Genera 7. The lack of examples also makes the process of porting version 6 code to Genera 7 a tedious affair. It is usually more efficient to just start from scratch.

**Summary** -- We would like to make it clear that we view the introduction of Genera 7 as a very positive step forward. It is only because we are so excited about the prospects of this system that we even bother to criticize it in any detail. Needless to say, this report does not cover all of our positive or negative feelings about presentations and Genera. We feel that if the system is going to evolve into something which will remain the preferred development environment in our research efforts for some time, that we must contribute as much as possible to the critique of this system. We would be happy to discuss this with the designers of the presentation system in more detail.