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, CO 80309-0430

In Proceedings of the International Conference on Intelligent Tutoring Systems,
Montreal, Canada, June 1988, pp 176-185

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.

Acknowledgements

The authors would like to thank our colleagues and students who have helped us to critically evaluate the usefulness of CRACK. We would like to thank especially Clayton Lewis and Raymond McCall for their criticism and suggestions. We are grateful to Sarah Reep and Maggie Boling who as professional kitchen designers have taken the time to collaborate with us on this project. Financial support for the work described in this paper was provided in part by grants from MCC and AT&T.

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, CO 80309-0430

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.

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.

Cooperative Problem Solving Systems

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 LISP-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.

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[1]. 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 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.

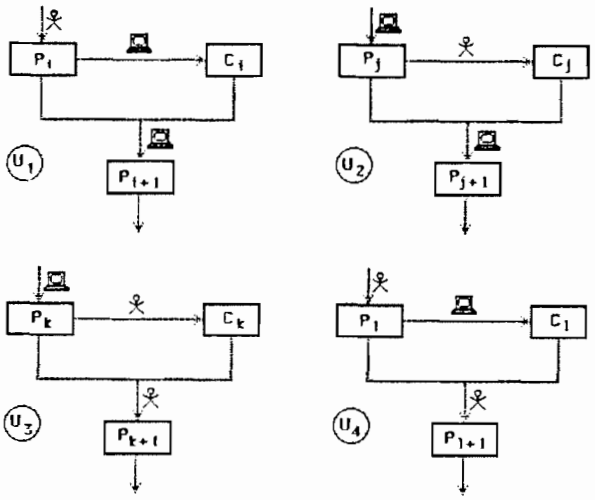
The Limitations of Construction Kits. Evaluating the Pinball and Music Construction Kits as prototypical examples against our objective of enhancing human problem-domain communication, we have found that their major shortcoming is that they do not assist the user in constructing interesting and useful 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.

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 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 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.

Human and computer play different roles in cooperative problem solving processes. In traditional expert systems (such as MYCIN and R1), the system plays the dominant role, and the user simply provides the necessary data, which are used by the system as a specification for deriving a suitable design design. This role assignment is easy to implement, but has turned out to be behaviorally unacceptable in many situations. In our research we are trying to support the full spectrum of cooperative problem solving processes (as illustrated by the diagrams U_1 to U_4), where the role assignments are determined by the nature of the task, the skill and knowledge level of the user and the decision of users, which role they prefer to play.

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

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 *satisfying* 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.

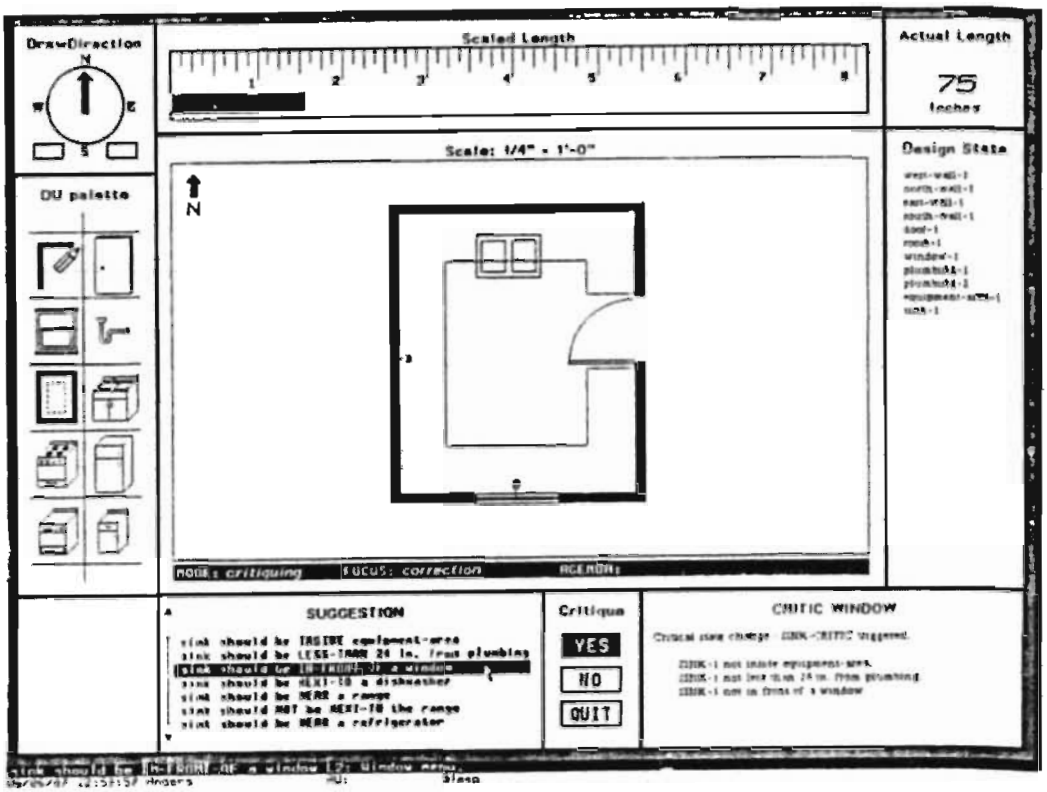


Figure 2: Suggestions from the SINK-CRITIC

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.

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

Crack: A Critic for Kitchen Design

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 2).

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.

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 showed that kitchen designers use design units at various levels of abstraction during a design process. First, the shape of the *equipment area* is determined, which is dependent upon the amount of usable wall length. Next, the various *work centers*[3] are considered, and then the appliances and cabinets which are part of the work centers are located. Finally, type and dimension of appliances and cabinets are specified.

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.

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.

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 earlier.

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 earlier, but can be modified by a user (see below).

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 2). 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). 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 4).

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-CRITC` 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 2).

The actual geometrical comparisons are performed by actions[4] defined on a pair of design units. For example `(defaction in-front-of equipment wall-fixture () ...)` defines an action on two generic design units `equipment` and `wall-fixture` (see Figure 3). 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.

Explanations

A user can ask for an explanation of each relation belonging to a critic (see Figure 5). 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.

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 knowledge to guide users, just like the Pinball Construction Kit. When critiquing is enabled, all critics are active. An in-

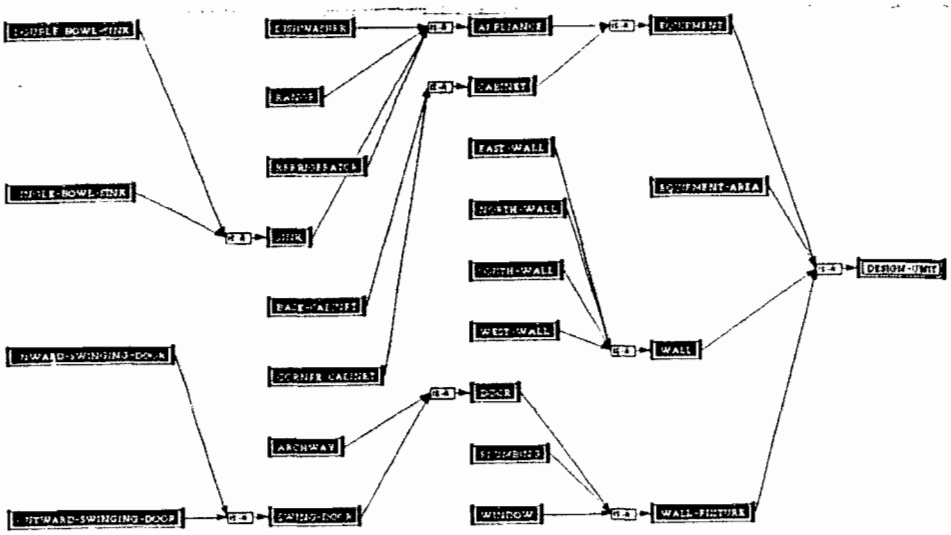


Figure 3: Design Unit Hierarchy

dividual 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 6.

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.

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 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.

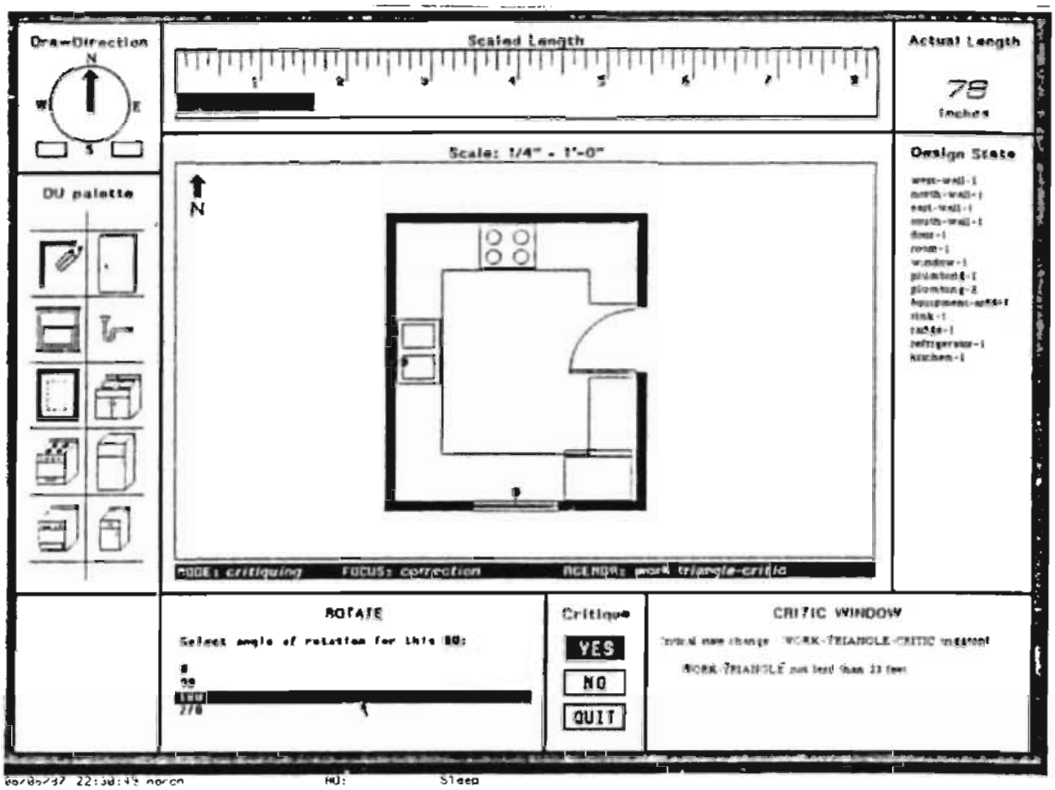


Figure 4: The Work-Triangle Critic -- A Critic for a Higher Level Concept

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. (See Figure 7.)

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 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 adviced 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 permanently (see previous section about the modification of the design knowledge). 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.

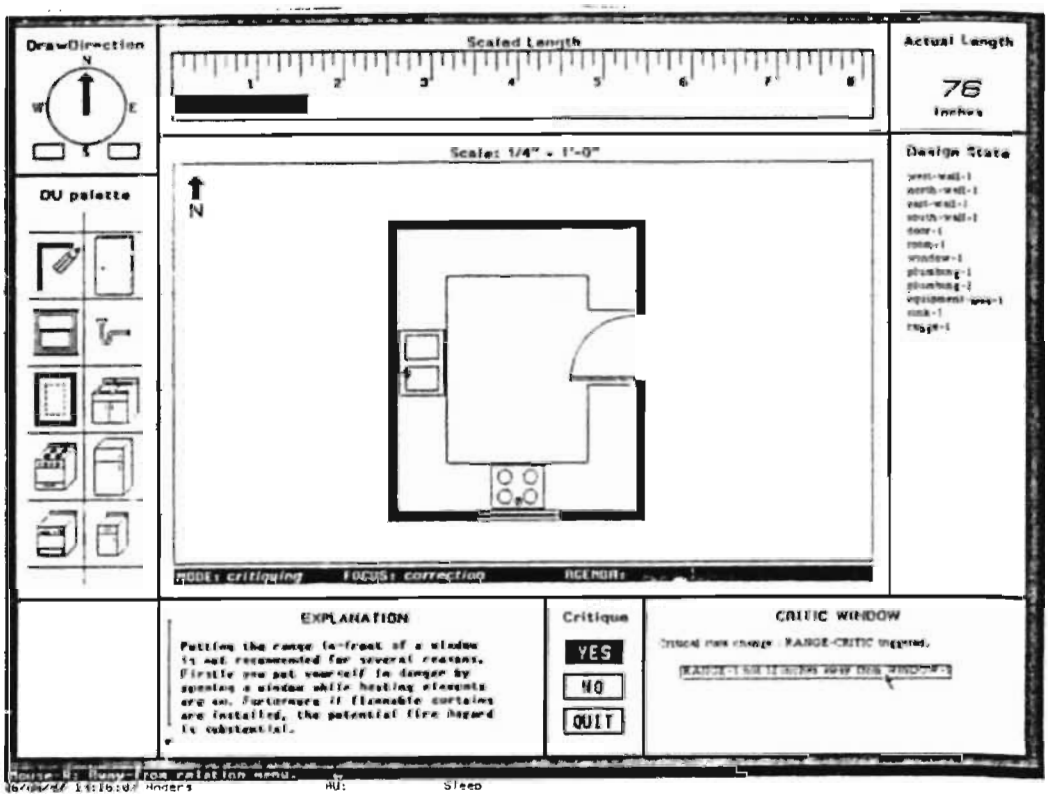


Figure 5: Explanation for AWAY-FROM Relation in RANGE-CRITIC

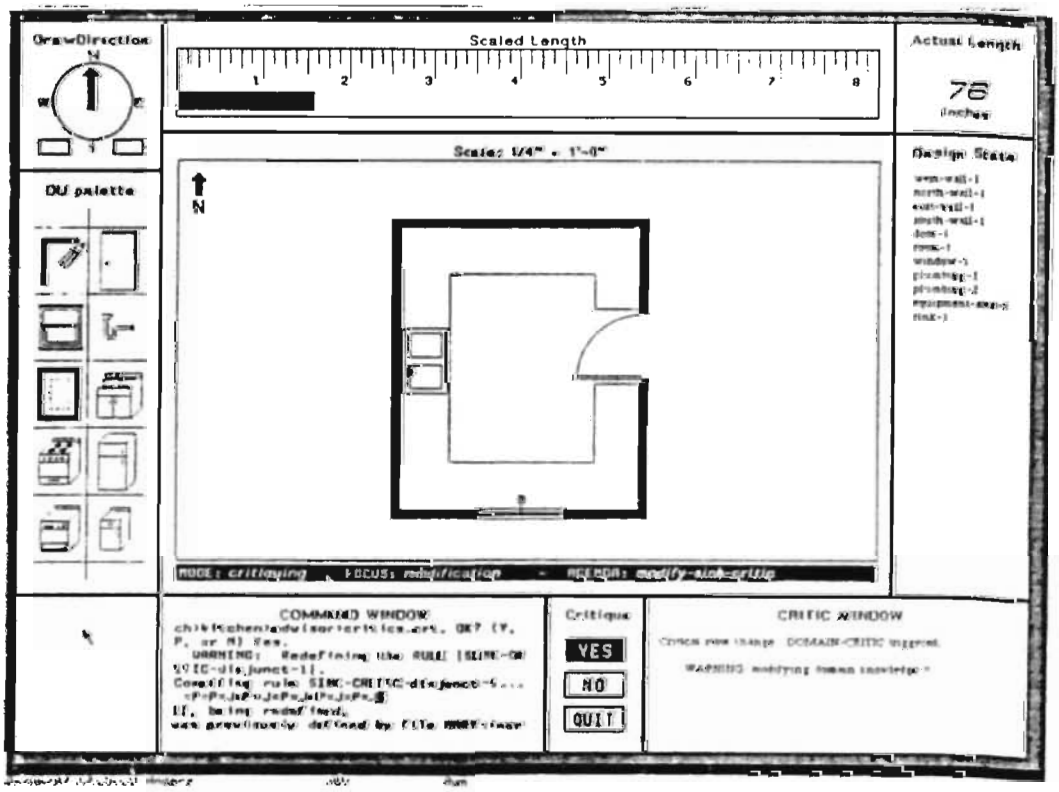


Figure 6: Modifying Design Knowledge

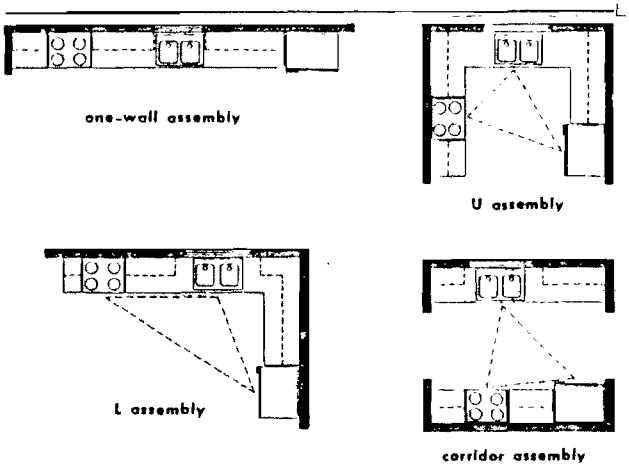


Figure 7: Four Prototypical Kitchen Solutions

Notes

[1]The specific design vocabulary for CRACK is represented as a set of icons in a palette (see Figure 2).

[2]The *work triangle* is the center front distance between the three appliances sink, range and refrigerator.

[3]A *work center* is an abstraction of several pieces of equipment, and the four main work centers are the cleanup-center, the cooking-center, the storage-center and the preparation-center.

[4]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.

Acknowledgements

The authors would like to thank our colleagues and students who have helped us to critically evaluate the usefulness of CRACK. We would like to thank especially Clayton Lewis and Raymond McCall for their criticism and suggestions. We are grateful to Sarah Reep and Maggie Boling who as professional kitchen designers have taken the time to collaborate with us on this project. Financial support for the work described in this paper was provided in part by grants from MCC and AT&T.

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, Vol. 4, No. 4, 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. 3, 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.