University of Colorado at Boulder

Department of Computer Science

ECOT 7-7 Engineering Center
Campus Box 430
Boulder, Colorado 80309-0430
(303) 492-7514

TO APPEAR IN THE PROCEEDINGS OF THE ACM CONFERENCE ON
HUMAN FACTORS IN COMPUTING SYSTEMS (CHI '90), SEATTLE, WN, APRIL 1 - 5, 1990

Using Critics to Empower Users

Gerhard Fischer, Andreas C. Lemke, and Thomas Mastaglio

Department of Computer Science and Institute of Cognitive Science
University of Colorado, Boulder, CO 80309-0430
303 492-1502, gerhard@boulder.colorado.edu

Anders I. Morch

NYNEX Artificial Intelligence Laboratory
White Plains, NY 10604
914 683-2209, anders@nynex-ms.nynexst.com

Abstract

We describe the critiquing approach to building knowledge-based interactive systems. Critiquing supports computer users in *their* problem solving and learning activities. The challenges for the next generation of knowledge-based systems provide a context for the development of this paradigm. We discuss critics from the perspective of overcoming the problems of high-functionality computer systems, of providing a new class of systems to support learning, of extending applications-oriented construction kits to design environments, and of providing an alternative to traditional autonomous expert systems. One of the critiquing systems we have built — JANUS, a critic for architectural design — is used as an example of the key aspects of the critiquing process. We also survey additional critiquing systems developed in our and other research groups.

KEYWORDS: critics, critiquing, high-functionality computer systems, intelligent support systems, design environments, cooperative problem solving systems.

Using Critics to Empower Users

Gerhard Fischer, Andreas C. Lemke, and Thomas Mastaglio

Department of Computer Science and Institute of Cognitive Science
University of Colorado, Boulder, CO 80309-0430
303 492-1502, gerhard@boulder.colorado.edu

Anders I. Morch

NYNEX Artificial Intelligence Laboratory
White Plains, NY 10604
914 683-2209, anders@nynex-ms.nynexst.com

ABSTRACT
We describe the critiquing approach to building knowledge-based interactive systems. Critiquing supports computer users in *their* problem solving and learning activities. The challenges for the next generation of knowledge-based systems provide a context for the development of this paradigm. We discuss critics from the perspective of overcoming the problems of high-functionality computer systems, of providing a new class of systems to support learning, of extending applications-oriented construction kits to design environments, and of providing an alternative to traditional autonomous expert systems. One of the critiquing systems we have built — JANUS, a critic for architectural design — is used as an example of the key aspects of the critiquing process. We also survey additional critiquing systems developed in our and other research groups.

KEYWORDS: critics, critiquing, high-functionality computer systems, intelligent support systems, design environments, cooperative problem solving systems.

INTRODUCTION

The critiquing approach is an effective way to make use of computer knowledge bases to aid users in their work and to support learning. Our experience with this approach includes several years of innovative system building efforts, the integration of cognitive and design theories, empirical observations and the evaluation of prototypes.

This paper combines our experience with the research efforts of others to articulate foundations and characteristics for the critiquing paradigm. We describe the rationale for critiquing and illustrate the approach using one of our systems (JANUS) as an example. A general characterization of the critiquing process is abstracted. Other systems are surveyed in terms of this critiquing framework. This shows the applicability and usefulness of critics across a variety of domains.

CHALLENGES FOR THE NEXT GENERATION OF KNOWLEDGE-BASED SYSTEMS

The next generation of knowledge-based systems will face the following challenges:

- They will be high-functionality systems, and their complete mastery will exceed most individual's cognitive capabilities.
- They will need to support a broad spectrum of learning and working activities.
- They should be integrated design environments.
- Rather than autonomous expert systems, they will often be joint human-computer systems supporting cooperative problem solving.

We will discuss how critics can meet each of these challenges in turn.

High-Functionality Computer Systems

As powerful computer hardware has become widely available, computers and computer-based systems for general applications have an increased range of capabilities. Technical complexity and the associated human cognitive costs to master these systems have both grown dramatically and limit the ability of users to take full advantage of them. One illustration of this situation is the Symbolics LISP machine; it contains over 30,000 functions and 3300 flavors (or classes) accompanied by 12 books with 4400 pages of written documentation.

Systems that offer a rich functionality are a mixed blessing. In a very large knowledge space, something related to what we need is likely to exist, but may be difficult to find. It is impossible and infeasible for any one individual to know such systems completely. Empirical studies [7] have shown that even very experienced users know only a subset of a large system. They encounter the following problems: they often do not know about the existence of building blocks and tools; they do not know how to *access* tools, or *when* to use them; they do not understand the *results* that tools produce, and they cannot combine, adapt, and modify tools according to their *specific* needs. Our goal is to increase the usability of high functionality computer systems, not by "watering down" functionality or steering the user toward only a subset of the systems' capabilities, but by facilitating learning about, access to, and application of the knowledge these systems contain. Critics contribute to these goals by bringing knowledge to bear when it is needed.

Systems to Support Learning

The computational power of high functionality computer systems can provide qualitatively new learning environments. Learning technologies of the future should be multi-faceted, supporting a spectrum extending from open-ended, user-centered environments such as LOGO [29] to guided, teacher-centered tutoring environments [35].

Tutoring is one way to first learn a new system. One can pre-design a sequence of microworlds and lead a user through them [1]. However, tutoring is of little help in supporting learning on demand when users are involved in their "own doing." Tutoring is not task-driven, because the total set of tasks *cannot* be anticipated. To support user-centered learning activities, we must build computational environments that match individual needs and learning styles. Giving users control over their learning and working requires that they become the initiators of actions and set their own goals.

In open learning environments users have unlimited control [29], but there are other problems. They do not support situations where users get stuck during a problem solving activity or settle at a suboptimal plateau of problem solving behavior. To successfully cope with new problems, users can benefit from a critic that points out shortcomings in their solutions and suggests ways to improve them.

In contrast to passive help systems, critics do not require users to formulate a question. Critics allow users to retain control; they interrupt only when users' products or actions could be improved. By integrating working and learning, critics offer users unique opportunities: to understand the purposes or uses for the knowledge they are learning; to learn by actively using knowledge rather than passively perceiving it, and to learn at least one condition under which the knowledge can be applied. A strength of critiquing is that learning occurs as a *natural byproduct* of the problem solving process.

Design Environments

To accomplish most things in this world, selective search, means-ends analysis, and other weak methods are not sufficient [32]; one needs to employ strong problem solving techniques with knowledge about the task domain. Designers (e.g., architects, composers, user interface designers, database experts, knowledge engineers) are experts in *their* problem domain. But domain specialists are not interested in learning the "languages of the computer;" they simply want to use the computer to solve their problems and accomplish required tasks. To shape the computer into a truly usable as well as useful medium, we have to make low-level primitives invisible. We must "teach" the computer the languages of experts by endowing it with the abstractions of application domains. This reduces the transformation distance between the domain expert's description of the task and its representation as a computer program. *Human problem-domain communication* is our term for this idea [13].

Design environments [25; 16] are tools that foster human problem-domain communication by providing a set of building blocks that model a problem domain. Design environments also incorporate knowledge about which components fit together and how. These systems contain critics that recognize suboptimal design choices and inefficient or useless structures.

Cooperative Problem Solving Systems

The goal of developing joint human-computer cognitive systems in which the computer is considered a cognitive amplifier has challenged the more widely understood goal of Artificial Intelligence: the understanding and building of autonomous, intelligent, thinking machines. A more important goal is to understand and build interactive knowledge media [34] or cooperative problem solving systems [11]. The major difference between classical expert systems, such as MYCIN and R1, and cooperative problem solving systems is in the respective roles of human and computer. Traditional expert systems ask the user for input, make all decisions, and then return an answer.

In a cooperative problem solving system, the user is an active agent and participates together with the system in the problem solving and decision making process. The precise roles played by the two parties depend on their different strengths with respect to knowledge of the goals and task domain. Critics are an important component of cooperative problem solving systems, especially when they are embedded in integrated design environments. These critics detect inferior designs, provide explanations and argumentation for their "opinion" and suggest alternative solutions.

Traditional expert systems are inadequate in situations where it is difficult to capture all necessary domain knowledge. Because they often leave the human out of the decision process, autonomous expert systems require a comprehensive knowledge base covering all aspects of the tasks to be performed; all "intelligent" decisions are made

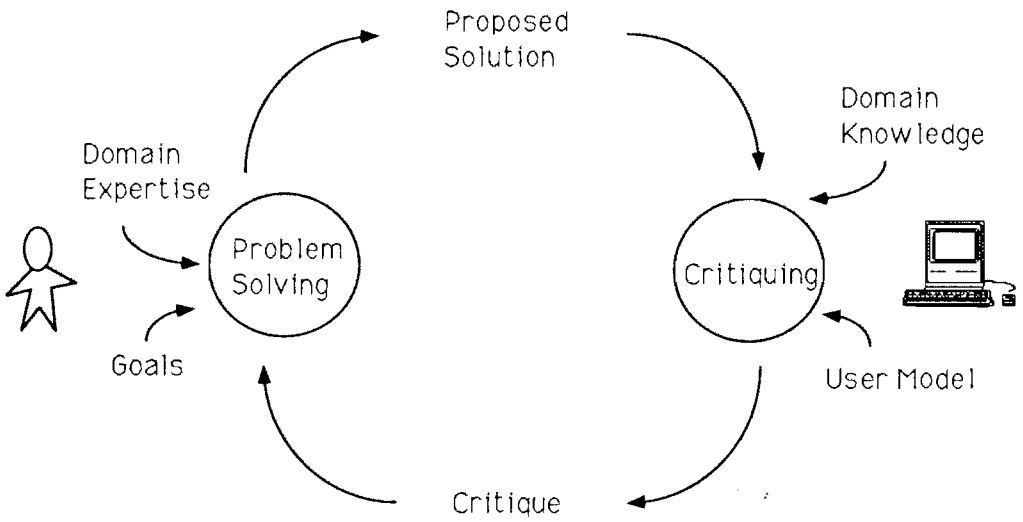

Figure 1: The Critiquing Approach

A critiquing system has two agents, a computer and a user, working in cooperation. Both agents contribute what they know about the domain to solving some problem. The human's primary role is to generate and modify solutions, while the computer's role is to analyze those solutions producing a critique for the human to apply in the next iteration of this process.

by the computer. Some domains, such as user interface design, are not sufficiently understood. Therefore, to create a complete set of principles that adequately capture knowledge in that domain is probably not possible. Other domains are so vast that tremendous effort is required to acquire all relevant knowledge. Critics are well suited to these situations because they need not be complete domain experts.

The traditional expert system approach is also inappropriate when the problem is ill-defined, that is, the problem cannot be precisely specified before a solution is attempted. In contrast, critics are able to function with only a partial task understanding.

THE CRITIQUING APPROACH

Critiquing is a way to present a reasoned opinion about a product or action (see Figure 1). The product may be a computer program, a kitchen design, a medical treatment plan; an action may be a sequence of keystrokes that corrects a mistake in a word processor document or a sequence of operating system commands.[1] An agent, human or machine, that is capable of critiquing in this sense is classified as a critic. Critics can consist of a set of rules or specialists for different issues; it is convenient to talk about each individual system component that reasons about a single issue as a critic.

Critics do not necessarily solve problems for users. The core task of critics is the recognition of deficiencies in a product and communication of those observations to users. Critics point out errors and suboptimal conditions that might otherwise remain undetected. Most critics make suggestions on how to improve the product. With this information users can fix the problems, seek additional advice or explanations.

Advisors [3] perform a function similar to critics except that they are the primary source for the solution. Users describe a problem, and they obtain a proposed solution from the advisor. In contrast to critics, advisors do not require users to present a partial or proposed solution to the problem.

JANUS: AN EXAMPLE

JANUS is a design environment based on the critiquing approach that allows designers to construct residential kitchens [16; 17]. The system enriches traditional design practice by augmenting the designer's creative and analytical skills. JANUS was developed as an integrated design environment to address some of the challenges of human problem-domain communication as previously discussed.

[1] In the remainder of the paper the term product is often used in a generic sense encompassing both product in a narrow sense and actions.

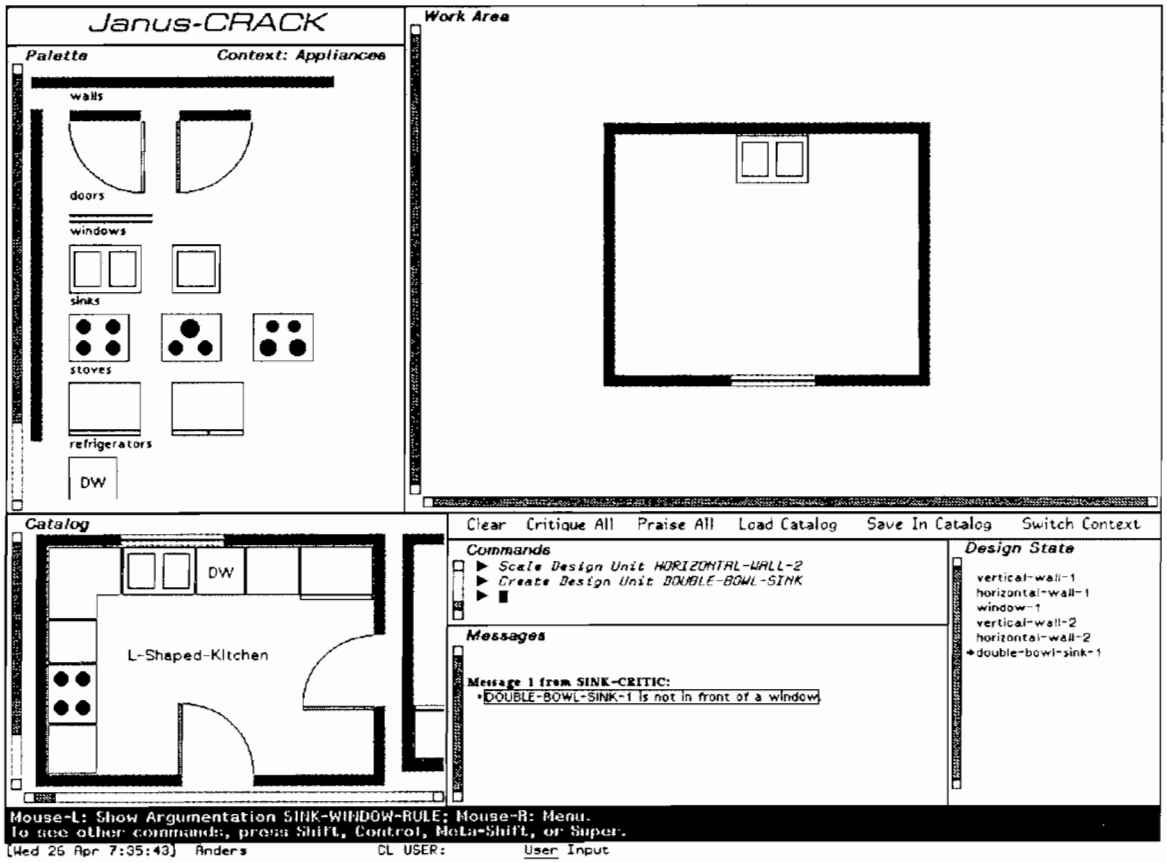

Figure 2: JANUS-CRACK: the `SINK-CRITIC`

JANUS-CRACK is the construction part of JANUS. Building blocks (design units) are selected from the *Palette* and moved to desired locations inside the *Work Area*. Designers can also reuse and redesign complete floor plans from the *Catalog*. The *Messages* pane displays critic messages automatically after each design change that triggers a critic. Clicking with the mouse on a message activates JANUS-VIEWPOINTS and displays the argumentation related to that message.

JANUS contains two subsystems: JANUS-CRACK and JANUS-VIEWPOINTS. JANUS-CRACK is a knowledge-based system supporting the construction of kitchens from domain-oriented building blocks called design units (Figure 2). JANUS-VIEWPOINTS is an issue-based hypertext system containing general principles of design to support argumentation about design. Integration of JANUS-CRACK and JANUS-VIEWPOINTS allows argumentation to resolve the problems that designers encounter during construction. JANUS is both a learning environment for design students and a tool for skilled designers.

JANUS-CRACK contains knowledge about how to distinguish "good" designs from "bad" designs and can explain that knowledge. The system knows how to combine building blocks into functional kitchens. Its knowledge includes three types of design principles [21]: building codes, such as *"the window area shall be at least 10% of the floor area."*, safety standards, such as *"the stove should not be installed under a window or within 12 inches of a window."*, and functional preferences, such as *"the work triangle should be less than 23 feet."*

Critics in JANUS-CRACK apply their design knowledge to critique the designer's partial solutions. They are implemented as condition-action rules, which are tested whenever the design is changed. The critics display messages, such as: *"sink not in front of a window"* in a critic window (see the Messages pane in Figure 2).

JANUS supports two design methodologies: design by composition (using the Palette) and design by modification (using the catalog). Examples in the catalog facilitate the redesign approach and can also be used to support learning. The user can copy both good and bad examples into the work area. One learning example is shown in Figure 3. The system can critique such designs to show how they can be improved, thus allowing users to learn from negative examples. To learn about good features of prestored designs, designers can run the `Praise All` command, thus getting positive feedback as well. Users can add their own designs to the catalog for future reuse or as additional learning examples. In addition to allowing changes to the design within the design environment, JANUS supports end user modification of the design environment itself [12].

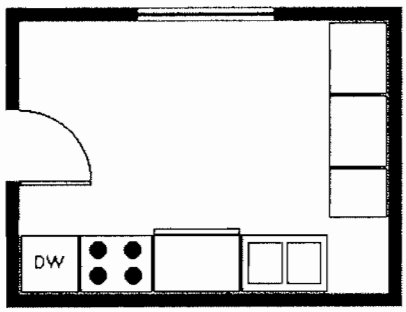

Figure 3: JANUS-CRACK: A Learning Example from the Catalog

The critics in JANUS detect the following suboptimal features of the kitchen shown in this figure: The width of the door is less than 36 inches, the dishwasher is not next to a sink, the stove is next to a refrigerator, the refrigerator is next to a sink, and the sink is not in front of a window.

THE PROCESS OF CRITIQUING

Figure 4 illustrates the subprocesses of critiquing: goal acquisition, product analysis, applying a critiquing strategy, explanation and advice giving. Not all of these processes are present in every critiquing system. This section describes these subprocesses and illustrates them with examples. JANUS does not illustrate all of the issues; therefore, we will refer occasionally to systems that are described in the Section DESCRIPTIONS OF CRITICS.

Goal Acquisition

Critiquing a product requires at least a limited understanding of the intended purpose of the product. That is problem knowledge which can further be separated into domain knowledge and goal knowledge. Just having domain knowledge without any understanding of the particular goals of the user, a critic can reason only about characteristics that pertain to all products in the domain. For example, domain knowledge allows JANUS to point out that stoves should not be placed in front of a window, because this arrangement constitutes a fire hazard. For a more extensive evaluation of a product, some understanding of the user's specific goals and situation is required.

A critic can acquire an understanding of the user's goals in several ways. Using an *implicit goal acquisition* approach, a general goal is built into the system. For example, JANUS is built for the problem domain of residential kitchen design, and the user's goal is assumed to be to design a "good" residential kitchen. Another approach is for the system to recognize goals by observing the evolving product constructed by users; this is goal recognition. A kitchen with a table and chairs located in the center of the kitchen suggests that the user intends to eat meals in the kitchen. Goal recognition presupposes solutions that approximate a solution to the user's problem. If the product fails to come close to the user's goal, the critic either cannot infer that goal or might infer one that is different from the user's. A critic may also have access to an explicit representation of the problem to be solved, for example users would communicate to the system that they need a kitchen for a large family.

Product Analysis

There are two general approaches to critiquing: *differential* and *analytical* critiquing. In the former approach, the system generates its own solution and compares it with the user's solution pointing out the differences. An advantage of differential critiquing is that all differences can be found. Some domains allow radically different, but equally valid solutions. This is a potential problem if the system generates its solution without regard to the user's solution approach. If user and system solutions differ fundamentally, the critic can only say that the system solution achieves good results but cannot explain why the user's solution is less than optimal.

Different solution attempts fulfill the goals to different degrees or are associated with different undesirable effects. In such situations, *metrics* are needed to measure the quality of alternative solutions [14]. Based on the controversial nature of design problems, alternative, conflicting metrics can be defined and have to be reconciled by negotiation and argumentation.

An *analytical critic* checks products with respect to predefined features and effects. Analytical critics identify suboptimal features using pattern matching [10], and expectation-based parsers [9]. In analytical approaches, critics do not need a complete understanding of the product. JANUS is an analytical critic that uses a set of rules to identify undesirable spatial relationships between kitchen design units. JANUS does not identify all possible problems within a kitchen design. Its rule base allows it to critique kitchens without knowing exact requirements and preferences of the kitchen user.

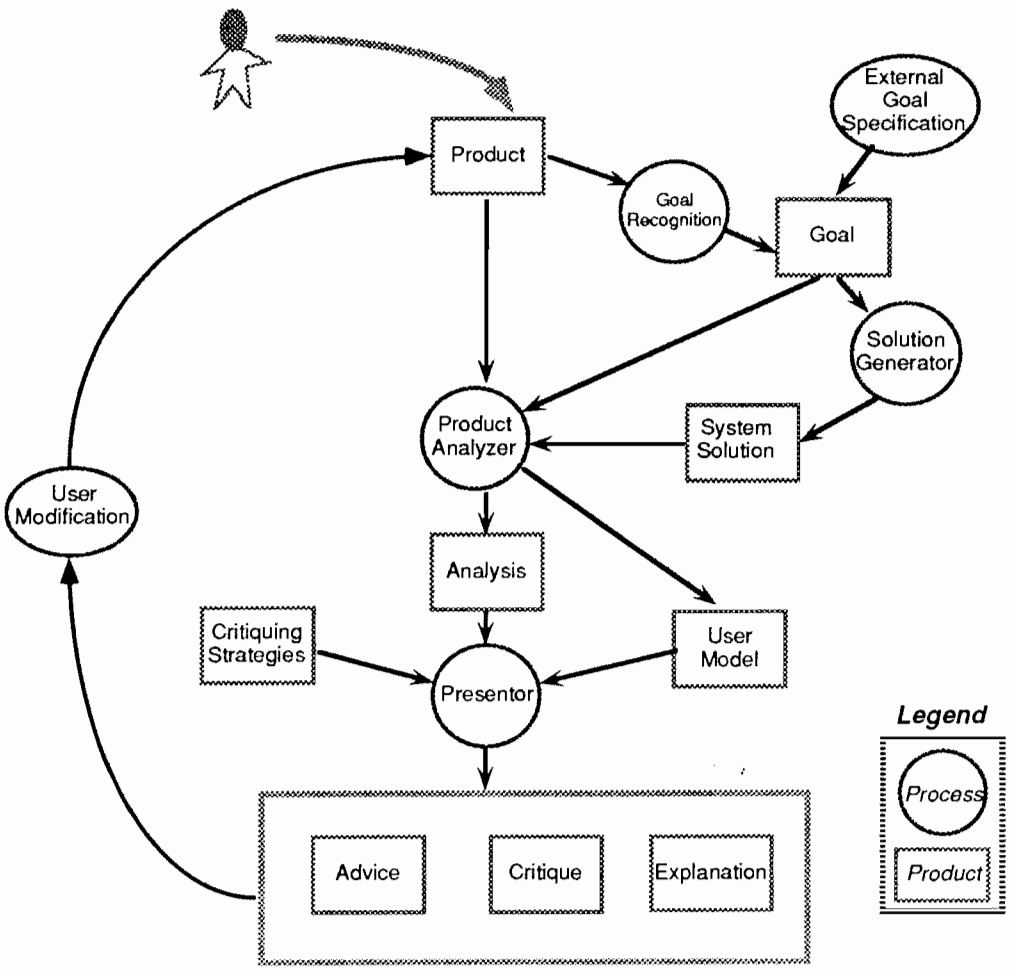

Figure 4: The Critiquing Process

Users initiate the critiquing process by presenting a product to the critic. In order to evaluate the product, the critic needs to obtain the user's goals either by recognizing them or from explicit user input. The product analyzer evaluates the product against the goal specification. Some critics do this by generating their own solution and comparing it to the user's. A presentation component uses the product analysis to formulate a critique, to give advice on how to make improvements, and to provide explanations. Critiquing strategies and a user model control the kind of critique, its form and timing. Based on the critique, the user generates a new version of the product, and the cycle repeats,integrating the new insight.

Critiquing Strategies

Critiquing strategies and a user model control the presentation component of a critic. The critiquing strategies determine what aspects to critique and when and how to intervene in the working process of the user. Critiquing strategies differ depending on the predominant use of the system, either to help users solve their problems or as a learning environment.

The user's perception of critics. Like recommendations from colleagues or co-workers, messages from a critic can be seen as helpful or hindering, as supportive of or interfering with the accomplishment of goals. Critiquing strategies should consider intrusiveness and the emotional impact on the user. *Intrusiveness* is the users' perception of how much the critiquing process is interfering with their work. Critics can either interfere too much or fail to provide sufficient help, depending on the frequency of feedback, the complexity of the tasks, and the sophistication of the user. *Emotional impact* relates to how users feel about having a computer as an intelligent assistant. Critiquing from a computer might be more tolerable than critiquing from humans if it is handled as a private matter between the human and the computer.

What should be critiqued? Educational critics, whose prime objective is to support learning, and performance critics, whose prime objective is to help produce better products, have different requirements for their critiquing strategies. A performance critic should help users create high-quality products in the least amount of time using as few resources as possible. Learning is not the primary concern of performance systems but can occur as a by-product of the interaction between user and critic. Educational critics should maximize the information users retain to improve their future performance.

Most performance critics (e.g., FRAMER, JANUS, ROUNDSMAN, KATE; see Section DESCRIPTIONS OF CRITICS) do not select specific aspects of a product to critique. They evaluate the product as a whole to achieve the highest possible quality. Some critics critique selectively based on a policy specified by the user. LISP-CRITIC (described in Section DESCRIPTIONS OF CRITICS), for example, operates differently depending on whether readability or machine efficiency is specified as the primary concern for writing LISP programs.

Educational critics (e.g., the WEST system [2]; also described in Section DESCRIPTIONS OF CRITICS) usually employ a more complex intervention strategy that is designed to maximize information retention and motivation by users. For example, an educational critic may forego an opportunity to critique when it occurs directly after a previous critiquing episode.

Most existing critics operate in the *negative* mode by pointing out suboptimal aspects of the user's product or solution. A *positive* critic recognizes the good parts of a solution and informs users about them. For performance critics, a positive critic helps users retain the good aspects of a product in further revisions, for educational critics, it reinforces the desired behavior and aids learning.

Intervention strategies. Intervention strategies determine when a critic should interrupt and how. *Active critics* exercise control over the intervention strategy by critiquing a product or action at an appropriate time. They function like active agents by continuously monitoring users, responding to individual user actions. *Passive critics* are explicitly invoked by users when they desire an evaluation. Passive critics usually evaluate the (partial) product of a design process, not the individual user actions that resulted in the product.

For active critics the intervention strategy must specify when to send messages to the user. Intervening immediately after a suboptimal or unsatisfactory action has occurred (an immediate intervention strategy) has the advantage that the problem context is still active in the users' mind, and they should remember how they arrived at the solution. The problem can be corrected immediately. A disadvantage of active critics is that they may disrupt cognitive processing and cause short term memory loss. Users then need to reconstruct the goal structure that existed before the intervention. Delayed critic messages may appear out of context and hence come too late to prevent the user from heading towards an undesirable state.

Critics can use any of various intervention modes that differ in the degree to which users' attention is attracted. A critic can force users to attend to the critique by not allowing them to continue with their work. A less intrusive mode is the display of messages in a separate critic window on the screen. This gives users a choice whether to read and process the message immediately or first complete an action in progress. The messages should be displayed in such a way that they do not go unnoticed. Those messages that pertain to users' current focus of attention should be easy to find rather than being hidden among a large set of messages related to other aspects of the product.

Adaptation Capability

To avoid repetitive messages and to accommodate different user preferences and users with different skills, a critiquing system needs an adaptation capability. A critic that persistently critiques the user on a position with which the user disagrees is unacceptable, especially if the critique is intrusive. A critic that constantly repeats an explanation that the user already knows is also unacceptable.

Critics can be adaptable or adaptive. Systems are called adaptable if the user can change the behavior of the system. An adaptive system is one that automatically changes its behavior based on information observed or inferred. An adaptation capability can be implemented by simply disabling or enabling the firing of particular critic rules, by allowing the user to modify or add rules, and by making the critiquing strategy dependent on an explicit, dynamically maintained user model.

User models in critics [14] share ideas and goals with student modeling in intelligent tutoring systems [4] and with similar efforts in advice giving natural language dialogue systems [23]. Computer critics require dynamic, persistent user models that can change over time but are accessible to the human user for inspection and modification. How to acquire and represent individual user models is a topic of ongoing research [26].

Explanation Capability

Critics have to be able to explain the reasons for their interventions. This provides users with an opportunity to assess the critique and then to decide whether to accept it. Knowing why a product was critiqued helps users to learn the underlying principles and avoid similar problems in the future. In a critiquing system, explanations can be focused on the specific differences between the system's and the user's solutions, or on violations of general guidelines. One particular approach that uses argumentation as the fundamental structuring mechanism for explanations in hypermedia format, is illustrated in the JANUS-VIEWPOINTS system [17].

Advisory Capability

All critics detect suboptimal aspects of the user's product (*problem detection mode*). Some critics require the user to determine how to improve the product by making changes to address the problems pointed out by the critic. Other critics, however, are capable of suggesting alternatives to the user's solution. We call these *solution-generating* critics. In the JANUS system, a simple problem detecting critic points out that there is a stove in front of a window. A solution-generating critic would, in addition, suggest a better location.

DESCRIPTIONS OF CRITICS

The purpose of this section is to provide an overview of critiquing systems that have influenced the development of the paradigm or that illustrate an interesting aspect of it. We first describe two critic systems developed in our laboratory (LISP-CRITIC and FRAMER). After that, we survey systems developed by others.

LISP-CRITIC

LISP-CRITIC [10; 15] is a system designed to support programmers. It helps its users to both improve the program they are creating and to acquire programming knowledge on demand. Programmers ask LISP-CRITIC for suggestions on how to improve their code. The system then suggests transformations that make the code more cognitively efficient (i.e., easier to read and maintain) or more machine efficient (i.e., faster or requiring less memory).

When LISP-CRITIC finds pieces of code that could be improved, it shows the user its recommendation. Users can accept the critic's suggestion, reject it, or ask for an explanation to aid in making that decision. For example, LISP-CRITIC suggests that the user replace a single conditional `cond` function with an `if` function. The user can request an explanation of why `if` is preferable to `cond`. The system develops an appropriate explanation, consulting a user model, and displaying the explanation in hypertext form. The user can use the explanation to access more detailed information available about LISP in an on-line documentation system (the Symbolics Document Examiner). To adequately support a wide range of user expertise, the system incorporates a user modeling component [26]. LISP-CRITIC uses that model to customize explanations so that they cover only what the user needs to know.

FRAMER

FRAMER [25] is an innovative design environment for developing program frameworks, components of window-based user interfaces on Symbolics LISP machines. The purpose of the FRAMER design environment is to enable designers to make use of a high-level abstraction — program frameworks — with little prior training.

FRAMER contains a knowledge base of design rules for program frameworks. The rules evaluate the completeness and syntactic correctness of the design as well as its consistency with the interface style used on Symbolics Lisp machines. The critics are either mandatory ones that represent absolute constraints that must be satisfied for program frameworks to function properly and optional critics that inform the user of issues that typically are dealt with differently. The critics are active, and the system displays the messages relevant to the currently selected checklist item in the window entitled *Things to take care of* (Figure 5). Each message is accompanied by up to three buttons: *Explain*, *Reject*, and *Execute*. The **`Explain`** button displays an explanation of the reasons why the designer should consider this critic suggestion; it also describes ways to achieve the desired effect. Optional suggestions have a **`Reject`** or **`Unreject`** button depending on the state of the suggestion. The **`Execute`** button accesses the advisory capability of FRAMER, which is available for issues that have a reasonable default solution.

A previous version of FRAMER employed a passive critiquing strategy. Experimental evidence [25] showed that users often invoked the critic too late when a major incorrect decision had already been made. The active strategy with continuous display of messages used in the newest version of FRAMER solved this problem. FRAMER prevents its users from permanently ignoring the critics by using the checklist. Checklist items cannot be checked off until all suggestions are either resolved or rejected.

Short Descriptions of Critics

What makes the critiquing approach attractive is that it has generality across a wide range of domains. Most critics have been developed as research vehicles, but a few are successful commercial applications. Critic or critic-like systems have been developed for the following application domains.

- *Education.* The WEST system, an early effort to build a computer coach [2], pioneered fundamental ideas that the critiquing paradigm incorporates. WEST builds a bridge between open learning environments and tutoring. Explicit intervention and teaching strategies are represented in the system and operate using information contained in a model of the user. WEST provided an early demonstration of how to construct an intelligent learning environment. Another system, that pioneered many current ideas in simulation-based learning environments, STEAMER, was later augmented with a critic. STEAMER/Feedback Mini-Lab [18] is an environment in which simulated devices, such as steam plant controllers, can be assembled and tested. After students have constructed a device, they may request a critique from the system.
- *Medical applications.* Researchers in the domain of medicine developed several of the early critiquing systems. These systems were developed to aid the physician in diagnosis and planning of patient treatment. Miller and colleagues at Yale Medical School have done the majority of the work in this area [28]. A version of ONCOCIN, an expert system for cancer therapy [24], also uses the critiquing approach. The ROUNDSMAN system [30] is a critic in the domain of breast cancer treatment. It bases its critique on studies in the medical literature.

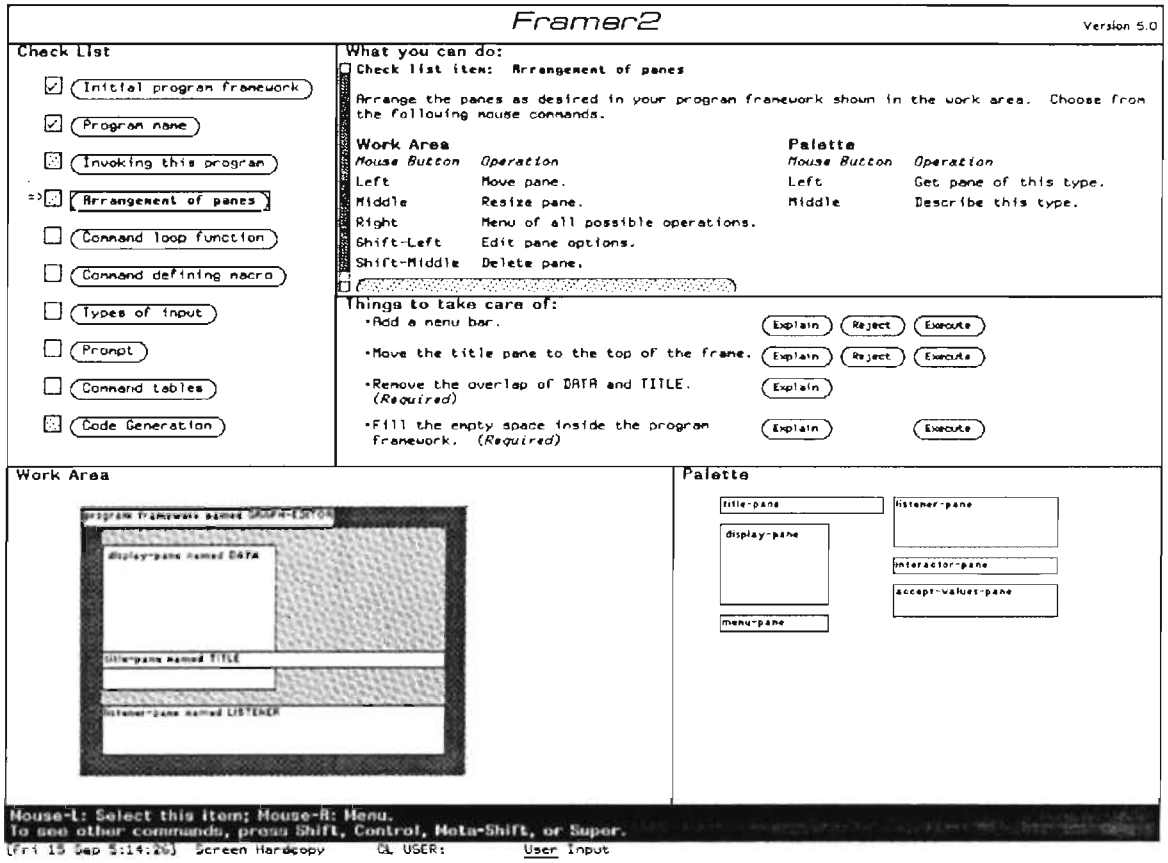

Figure 5: FRAMER

This figure shows a screen image of a session with FRAMER. The system has the following components. The checklist describes the elements of the task of designing a program framework. The *What you can do* window shows the detailed options pertaining to a checklist item. The window entitled *Things to take care of* displays the critic messages. The work area is the place were frameworks are assembled in a direct manipulation interaction style. A palette contains title panes, display panes, and other primitive parts for constructing program frameworks. FRAMER also offers a catalog (not shown) for design by modification.

- *Circuit design.* An early digital circuit design system that incorporated a critiquing approach is CRITTER [22]. The Design Advisor ™ [33], is a successful commercial system developed at NCR that provides advice on application-specific, integrated circuit designs. Voltaville, developed at LRDC, University of Pittsburgh [20], is a prototype discovery environment for circuits. It is designed to build scientific inquiry skills.
- *Decision-making.* DecisionLab [31] applies the critiquing approach to coach users in managerial decision making. DECAD [27] watches over the shoulder of the decision maker, interjecting advice or a critique when appropriate.
- *Authoring.* Critiquing has proven to be a successful approach to providing assistance to those users engaged in authoring tasks. WANDAH [19] uses critiquing to assist authors in all phases of writing; it is now commercially available for personal computers as "HBJ Writer." ACTIVIST [14] is an active help system for a screen-oriented editor. One knowledge-based system provides assistance to teachers doing curriculum development [36].
- *Computer usage.* WIZARD is an active help system for users of the VMS operating system [9]. PROLOG Explaining [5] critiques a user's explanation of PROLOG code to guide the user toward a better understanding of the PROLOG language. The GRACE Project at the NYNEX Artificial Intelligence Laboratory [6] is developing a multi-faceted integrated learning environment for COBOL programming. It consists of a critic, a tutor, and a hypertext system. KATE [8] critiques software specifications for automated library systems.

CONCLUSION

Critiquing is an emerging approach to building knowledge-based systems. Critics are a major component of cooperative problem solving systems, which can serve both as performance systems that help users solve real-world problems and as learning environments that support incremental learning. A strength of critics is that they draw on the potential of human problem solvers where appropriate. Critics can operate with various degrees of domain knowledge, and can be upgraded gradually to give more and more comprehensive support. However, critiquing is not without its limitations. Supporting users in their own doing means that details of user goals are often not available to the system, limiting the specificity of the critique the system can provide. Overall, the critiquing paradigm is an effective approach for applying knowledge-based system technology to empower users of computer-based systems in a broad range of domains.

ACKNOWLEDGMENTS

Many people have contributed over the last decade to the development of our notion of the critiquing paradigm. The authors would like to thank especially: the members of the Janus Design Project (Ray McCall, Kumiyo Nakakoji, and Johnathan Ostwald), the members of the LISP-CRITIC project (Heinz-Dieter Boecker, Chris Morel, Brent Reeves, and John Rieman), all the people who have participated in discussions about the general framework for critiquing (Thomas Schwab, Helga Nieper-Lemke, Curt Stevens, Tom DiPersio, and Hal Eden), and the HCC research group as a whole. This research was partially supported by grant No. IRI-8722792 from the National Science Foundation, grant No. MDA903-86-C0143 from the Army Research Institute, and grants from the Intelligent Interfaces Group at NYNEX and from Software Research Associates (SRA), Tokyo.

REFERENCES

1. J.R. Anderson, B.J. Reiser. "The LISP Tutor". *BYTE 10*, 4 (April 1985), 159-175.

2. R.R. Burton, J.S. Brown. An Investigation of Computer Coaching for Informal Learning Activities. In *Intelligent Tutoring Systems*, D.H. Sleeman, J.S. Brown, Eds., Academic Press, London - New York, 1982, ch. 4, pp. 79-98.

3. J.M. Carroll, J. McKendree. "Interface Design Issues for Advice-Giving Expert Systems". *Communications of the ACM 30*, 1 (January 1987), 14-31.

4. W.J. Clancey. "Qualitative Student Models". *Annual Review of Computing Science 1* (1986), 381-450.

5. M.J. Coombs, J.L. Alty. "Expert Systems: An Alternative Paradigm". *International Journal of Man-Machine Studies 20* (1984).

6. S. Dews. Developing an ITS in a Corporate Setting. Proceedingsof the Human Factors Society 33rd Annual Meeting, Volume 2, Human Factors Society, 1989, pp. 1339-1342.

7. S.W. Draper. The Nature of Expertise in UNIX. Proceedings of INTERACT'84, IFIP Conference on Human-Computer Interaction, Amsterdam, September, 1984, pp. 182-186.

8. S. Fickas, P. Nagarajan. "Critiquing Software Specifications". *IEEE Software 5*, 6 (November 1988), 37-47.

9. T.W. Finin. Providing Help and Advice in Task Oriented Systems. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 1983, pp. 176-178.

10. G. Fischer. A Critic for LISP. Proceedings of the 10th International Joint Conference on Artificial Intelligence (Milan, Italy), Los Altos, CA, August, 1987, pp. 177-184.

11. G. Fischer. "Communications Requirements for Cooperative Problem Solving Systems". *The International Journal of Information Systems Special Issue on Knowledge Engineering* (1990). to be published.

12. G. Fischer, A. Girgensohn. End-User Modifiability in Design Environments. Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA), ACM, New York, April, 1990.

13. G. Fischer, A.C. Lemke. "Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication". *Human-Computer Interaction 3*, 3 (1988), 179-222.

14. 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.

15. G. Fischer, T. Mastaglio. Computer-Based Critics. Proceedings of the 22nd Annual Hawaii Conference on System Sciences, Vol. III: Decision Support and Knowledge Based Systems Track, IEEE Computer Society, January, 1989, pp. 427-436.

16. G. Fischer, R. McCall, A. Morch. Design Environments for Constructive and Argumentative Design. Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX), ACM, New York, May, 1989, pp. 269-275.

17. G. Fischer, R. McCall, A. Morch. JANUS: Integrating Hypertext With a Knowledge-Based Design. Proceedings of Hypertext'89, ACM, November, 1989, pp. 105-117.

18. K. Forbus. An Interactive Laboratory for Teaching Control System Concepts. Report 5511, BBN, Cambridge, MA, 1984.

19. M.P. Friedman. WANDAH - A Computerized Writer's Aid. In *Applications of Cognitive Psychology, Problem Solving, Education and Computing*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, Chap. 15, pp. 219-225.

20. R. Glaser, K. Raghavan, L. Schauble. Voltaville: A Discovery Environment to Explore the Laws of DC Circuits. Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada), June, 1988, pp. 61-66.

21. R.J. Jones, W.H. Kapple. *Kitchen Planning Principles - Equipment - Appliances.* Small Homes Council - Building Research Council, University of Illinois, Urbana-Champaign, IL, 1984.

22. V.E. Kelly. The CRITTER System: Automated Critiquing of Digital Circuit Designs. Proceedings of the 21st Design Automation Conference, 1985, pp. 419-425.

23. A. Kobsa, W. Wahlster (Ed.). *User Models in Dialog Systems.* Springer-Verlag, New York, 1989.

24. C. Langlotz, E. Shortliffe. "Adapting a Consultation System to Critique User Plans". *International Journal of Man-Machine Studies 19* (1983), 479-496.

25. A.C. Lemke. *Design Environments for High-Functionality Computer Systems.* Ph.D. Th., Department of Computer Science, University of Colorado, Boulder, CO, July 1989.

26. T. Mastaglio. User Modelling in Computer-Based Critics. Proceedings of the 23rd Hawaii International Conference on the System Sciences, IEEE Computer Society, 1990. to be published.

27. F. Mili. A Framework for a Decision Critic and Advisor. Proceedings of the 21st Hawaii International Conference on System Sciences, Vol III, Kailu-Kona, Hawaii, January 5-8, 1988, Jan, 1988, pp. 381-386.

28. P. Miller. *Expert Critiquing Systems: Practice-Based Medical Consultation by Computer.* Springer-Verlag, New York - Berlin, 1986.

29. S. Papert. *Mindstorms: Children, Computers and Powerful Ideas.* Basic Books, New York, 1980.

30. G.D. Rennels. *Lecture Notes in Medical Informatics.* Volume 32: *A computational model of reasoning from the clinical literature.* Springer Verlag, 1987.

31. J. Schiff, J. Kandler. Decisionlab: A System Designed for User Coaching in Managerial Decision Support. Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada), June, 1988, pp. 154-161.

32. H.A. Simon. "Whether Software Engineering Needs to Be Artificially Intelligent". *IEEE Transactions on Software Engineering SE-12*, 7 (July 1986), 726-732.

33. R.L. Steele. Cell-Based VLSI Design Advice Using Default Reasoning. Proceedings of 3rd Annual Rocky Mountain Conference on AI, Rocky Mountain Society for Artificial Intelligence, Denver, CO, 1988, pp. 66-74.

34. M.J. Stefik. "The Next Knowledge Medium". *AI Magazine 7*, 1 (Spring 1986), 34-46.

35. E. Wenger. *Artificial Intelligence and Tutoring Systems.* Morgan Kaufmann Publishers, Los Altos, CA, 1987.

36. K. Wipond, M. Jones. Curriculum and Knowledge Representation in a Knowledge-Based System for Curriculum Development. Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada), June, 1988, pp. 97-102.