University of Colorado at Boulder

Gerhard Fischer
Department of Computer Science

ECOT 7-7 Engineering Center
Campus Box 430
Boulder, Colorado 80309-0430
(303) 492-1502, FAX: (303) 492-2844
e-mail: gerhard@boulder.colorado.edu

IN "MODELING CREATIVITY AND KNOWLEDGE-BASED CREATIVE DESIGN", J. S. GERO AND M. L. MAHER (EDS), LAWRENCE ERLBAUM ASSOCIATES, HILLSDALE, NJ, 1992, PP. 235-258.

Creativity Enhancing Design Environments

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

Abstract. Computers have the potential to be creativity enhancing tools. But most of the current systems have not lived up to these expectations -- they have restricted rather than enhanced creativity. Designers were forced to express their goals, ideas and (partial) solutions at levels that were too remote from the problem domains they were dealing with.

To overcome these limitations, we have developed a conceptual framework and prototypical systems that allow designers to work with personal meaningful operations. Beyond providing domain-specific abstractions, our knowledge-based design environments can evaluate and criticize an evolving design and provide feedback to the designer. They integrate constructive and argumentative components. Support for end-user modifiability allows designers to extend the environments themselves. Knowledge-based design environments turn the computer into an *invisible instrument* and support *cooperative problem solving between the human designer and the computer.* Our experience with these systems demonstrates that they have the potential to serve as important stepping stones towards creativity enhancing environments.

Keywords. creativity enhancing environments, knowledge-based systems, human problem-domain communication, design environments, critics, design by composition, design by modification, domain-specific support for computer assisted design, end-user modifiability, integration of construction and argumentation, expert systems, cooperative problem solving systems

Creativity Enhancing Design Environments

Gerhard Fischer
Department of Computer Science and Institute of Cognitive Science
University of Colorado
Boulder Colorado 80309
USA

Abstract. Computers have the potential to be creativity-enhancing tools. But most of the current systems have not lived up to these expectations—they have restricted rather than enhanced creativity. Designers were forced to express their goals, ideas, and (partial) solutions at levels that were too remote from the problem domains they were dealing with.

To overcome these limitations, we have developed a conceptual framework and prototypical systems that allow designers to work with personally meaningful operations. Beyond providing domain-specific abstractions, our knowledge-based design environments can evaluate and criticize an evolving design and provide feedback to the designer. They integrate constructive and argumentative components. Support for end-user modifiability allows designers to extend the environments themselves. Knowledge-based design environments turn the computer into an *invisible instrument* and support *cooperative problem solving between the human designer and the computer.* Our experience with these systems demonstrates that they have the potential to serve as important stepping stones towards creativity enhancing environments.

1. Introduction

Tools (either conceptual ones or physical ones) have played a major role for human beings in scientific discoveries, problem solving capabilities, and in the power to design and create. Good tools provide only the necessary requirements and although they are not sufficient by themselves, without them creativity is severely limited. The computer has been seen by many as the ultimate tool to support creativity. But the achievements so far have fallen short of these expectations.

In this paper I first discuss a conceptual framework for creativity and enumerate a set of requirements for enabling creativity. Then I explore why there are so few computational environments for enabling and supporting creativity. Innovative system building efforts (construction kits and design environments) as steps towards creativity enhancing environments are then described and evaluated against the conceptual framework.

2. Creativity and Computers

2.1 Creativity

Creativity can be informally defined [Hayes 78] as consisting of acts that have some valuable consequence and that are novel or surprising. It is a special kind of problem-solving thinking requiring high motivation and persistence. The problem itself (as initially posed) is vague and ill-defined, so that part of the task is to formulate the problem itself. Creativity is often associated with divergent production abilities (i.e., doing tasks in which many different responses to the same situation must be generated) and less with convergent production abilities (i.e., tasks in which a person is expected to generate a single correct

answer).

Design [Simon 81] is one of the most promising activities to study creativity, based on the following attributes of design problems: Designers who tackle the same problem are likely to come up with different solutions [Jacob 77], good designers break rules all the time, design deals with ill-structured [Simon 73] and wicked problems [Rittel 72] (i.e., problems that are intrinsically open-ended, situation specific and controversial), and in design there are no optimal solutions, only trade-offs.

2.2 How can creativity be enhanced?

Hayes [Hayes 78] states the obvious that "there is no procedure which will guarantee that a person will invent something important or initiate a new artistic movement". The advice "Be more creative!" is about as helpful for a problem solver as saying to a software engineer "Think more clearly!". But software engineers in the 1980's are writing more complex and more interesting programs as their predecessors in the 1960's. High-school children write programs today that may have earned them a PhD twenty years ago. New technologies (e.g., better hardware, powerful programming environments) and new methodologies (e.g., structured programming, object-oriented programming, reuse and redesign) have enhanced the problem-solving power of the programmer. Can creativity be enhanced in similar ways in other computer-supported design tasks with the right tools and environments?

In the following, I enumerate requirements and techniques that have the potential to increase our chances of being more creative.

Developing a knowledge base. Having relevant knowledge does not guarantee creativity, but it is one important condition for it. Experts (uniformly across different domains, e.g., in chess, physics, painting, composing, etc.,) have at least 50,000 chunks about the relevant knowledge in their domain [Simon 81].

Create the right environment for creativity. Instead of putting all of our knowledge "into the head", some of this knowledge can be put "into the world" [Norman 88]. Ideas generation benefits from being reminded of the right thing at the right time, or being forced to argue for a position.

Look for analogies and the impact of representation on problem difficulty. Seeing a problem from a different perspective often helps to turn a difficult problem into an easy one. The crucial point is the switch from searching *within* a problem space to searching *for* a problem space. The *mutilated checker board* problem becomes trivial when it is seen as the *matchmaker problem* [Hayes 78; Newell, Simon 75; Kaplan,Simon 89]. In the "mutilated checker board problem" a checkerboard with 64 squares and a set of 32 rectangular dominoes are given. Each of the dominoes covers exactly 2 checkerboard squares. Obviously, the 32 dominoes can be arranged to cover the board completely. Now suppose that two (black) squares were cut from the opposite corners of the board. Can the remaining 62 squares of the board be covered using exactly 31 dominoes? The "matchmaker problem" takes place in a small village in the Midwest where 32 bachelors and 32 unmarried women live. Through tireless efforts, the village matchmaker succeeded in arranging 32 highly satisfactory marriages. Then one Saturday night, two drunken bachelors fatally stabbed each other. Can the matchmaker, through some quick arrangements, come up with 31 satisfactory marriages among the 62 survivors?.

Supporting the incremental unfolding of design spaces. In most design tasks, the design space only unfolds as designers work in it. Many requirements emerge only in the course of the design process,

when partial design solutions provide enough context to realize which issues are really important. This situation is aptly characterized by [Simon 81]: *"Architecture can almost be taken as a prototype for the process of design in a semantically rich task domain. The emerging design is itself incorporated in a set of external memory structures: sketches, floor plans, drawings of utility systems, and so on. At each stage in the design process, the partial design reflected in these documents serves as a major stimulus for suggesting to the designer what he would attend to next. This direction to new subgoals permits in turn new information to be extracted from memory and reference sources and another step to be taken toward the development of the design".* In our work we have demonstrated the relevance of supporting incremental approaches to design and problem solving with a system which allows users to incrementally formulate and reformulate a query in an information retrieval task [Fischer, Nieper-Lemke 89].

Reuse and redesign. If computational media live up to their name, namely that software is truly "soft", then they offer unique possibilities for reuse and redesign. Future developments should be directed towards the goal that programmability and end-user modifiability will allow that professionally produced items become changeable, adaptable, fragmentable, and quotable in ways that present software is not. Not only would professionals be able to construct grand images but others would be able to *reconstruct* personalized versions of these same images [diSessa, Abelson 86]. Another important argument for reuse and redesign is that complex systems evolve faster if there are stable subsystems.

Exploiting what people already know. A crucial element for increasing the competence and the independence of designers is to build environments that exploit what designers already know or that allow them to build natural bridges to their existing knowledge. For example, to support a spatial metaphor is so important, because we all know how to move around in space. Our concept of *human problem domain communication* (see section 3.1) is relevant for this goal, because it allows designers to work with abstractions of their *own* domain of expertise.

Explanation and argumentation. If knowledge with respect to the problem to be solved is "in the world," and partial designs serve as stimulus for further action, then we have to understand these partial design and their rationale. If the partial designs are embedded in design environments, then these environments should provide explanations and argumentations for them. Explanations and argumentation assist us in understanding someone else's rationale. They have the potential to enhance creativity, because they allow us to increase our knowledge base and to reason against another opinion. They can stimulate critical thinking, because every reason given is ammunition for the different people involved to argue about. Both help us to rethink our own implicit assumptions, enabling us to notice flaws in our own thoughts or to discover alternative ways to think about a problem.

Taking care of low-level clerical details. Environments are needed that allow designers to think and work on the important parts of problems. Designers should not worry about low-level details that are far removed from their actual objectives. The UNIX writer's workbench tools [Cherry 81] allow writers to focus on their ideas and not spend most of the time with spelling and style issues. Our LISP-CRITIC [Fischer 87a] frees LISP programmers from worrying about details of the coding activity, giving them more time to concentrate on their problem-solving activity.

Affection and appropriation. Papert [Papert 86] convincingly argues that one of the most important aspects of engaging people in creative acts is that they must "fall in love with what they are doing." They must make activities their own and they must be able to care about their work, a principle that he calls

"appropriation." To achieve affection and appropriation is more likely if designers can engage in activities in which they are truly interested and if they learn and do something which is of immediate benefit to them. People learn things best if they can use them—not sometime in the future but to solve a problem which they face right at this moment.

With this informal characterization of creativity, I will investigate the question, why there have been so few creative solutions in computational media and will then discuss knowledge-based design environments in the next section as steps towards creativity enhancing environments.

2.3 Why have there been so few Creative Solutions in Computational Media?

The summary answer to this question is that current computer systems inadequately support the issues enumerated in the previous section. In certain cases computational media of the past were just too impoverished to support creative expression. As long as

COMPUTER OUTPUT LOOKED LIKE THIS,

there was little room for creativity. Over the last few years, first daisy wheel printers started producing output that it was called "typewriter-quality." Nowadays laser printers allow *italic*, **bold**, and other typographic niceties, such as sub$_{\text{scripting}}$ and super$^{\text{scripting}}$. These technological capabilities gave people control over features with which they had no experience. Because few support tools and heuristics were provided, the set of new features led to the problem [Bentley 86] that

"Powerful *tools* can SOMETIMES be **powerfully abused!!**

Creativity is more than using any existing feature. High functionality computer systems (with tens of thousands tools embedded in them) have to provide guidance to the creative use of their features [Lemke, Fischer 90]. The situation changed from having weak generators (e.g., providing only upper-case letters) for problem solutions which gave no room for creativity to having powerful generators without support for selectivity and critiquing.

Another major reason for limiting creativity is that to do anything interesting, computers often require too much time and too much effort. Imagine the task of designing a pinball machine on your computer by writing statements in a programming language. The transformation distance between the goal state (the pinball machine) and the start state (a programming language) is too large. This prevents designers from thinking about the right kind of problems (e.g., how to locate bumpers, how to create a scoring scheme, etc.), and from getting something interesting done in a reasonable time.

The Music and PinBall Construction Kit (two interesting programs for the Apple MacIntosh from Electronic Arts; see Figures 2-1 and 2-2) provide domain-level building blocks (e.g., notes, sharps, flats, bumpers, flippers, etc.) to build artifacts in the domain of music and pinball machines. Designers can interact with the system in terms drawn from the problem domain; they need not learn abstractions peculiar to a computer system.

Our empirical investigations have shown that users familiar with the problem domain but inexperienced with computers had few problems using this system, whereas computer experts unfamiliar with the problem domain were unable to exploit its power. Most people considered it a difficult (if not impossible) task to achieve the same results using only the basic Macintosh system without the construction kit.

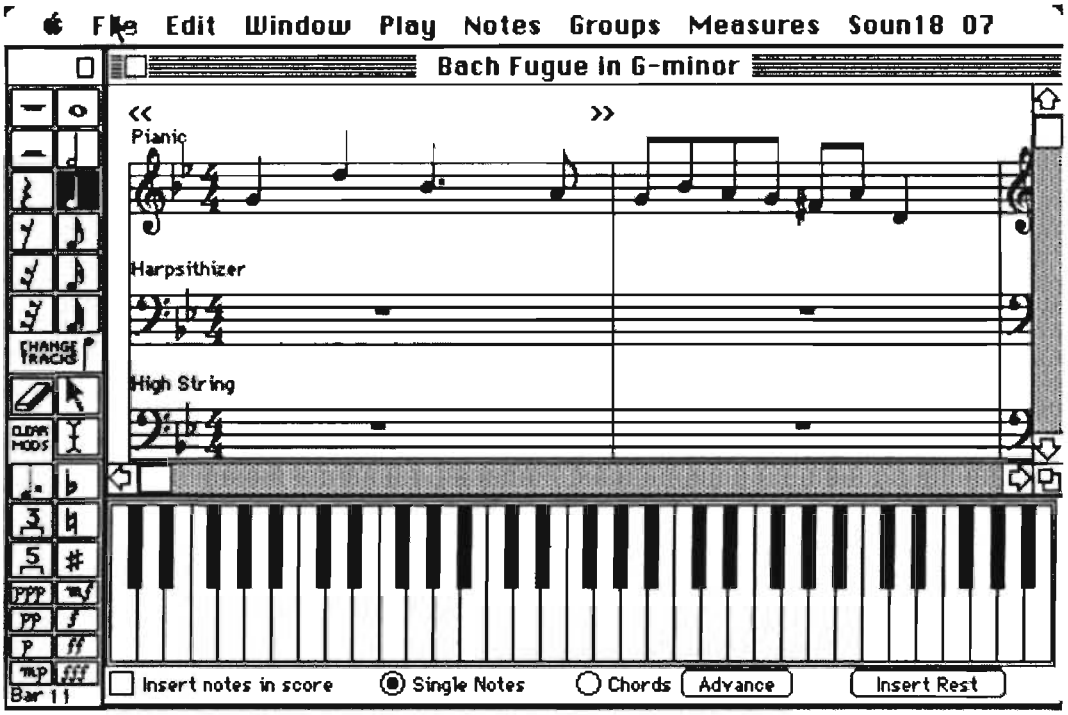

Figure 2-1: A Screen Image from the Music Construction Kit

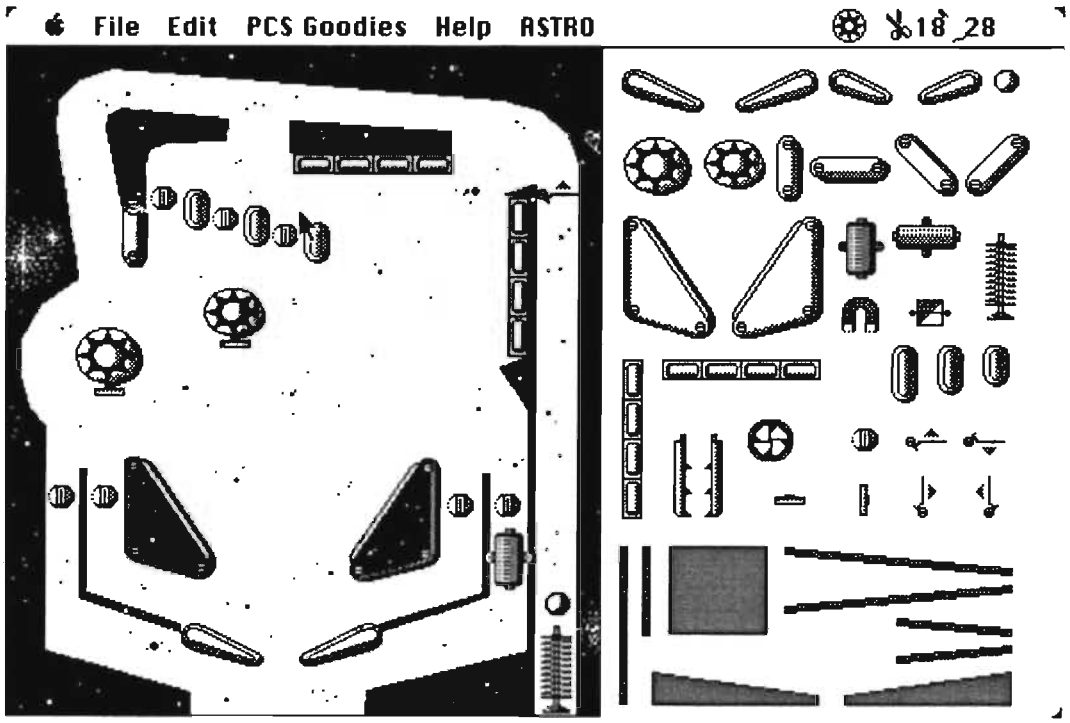

Figure 2-2: A Screen Image from the Pinball Construction Kit

Using the construction kit, our subjects experienced a sense of accomplishment, because they were creating their own impressive version of something that works (at least to a certain extent), yet is not difficult to make. By reducing the transformation distance between problem domain and system space, people were able to spend most of their time on interesting problems. They produced an interesting artifact quickly and were able to change it with little effort, supporting an additional requirement for creativity: "To come up with a great idea, one must have and explore many ideas."

As long as computational environments require substantial knowledge of computers, domain experts will be limited in using the computer as a creative medium. Another possibility would be that software engineers become the creative designers? This also has its limitations, because computer systems model more and more real world domains requiring substantial amount of domain-oriented knowledge. This was revealed in a recent empirical study [Curtis, Krasner, Iscoe 88], which showed that a major limitation in software engineering is *the thin spread of application knowledge among software engineers*. In the long run, it seems a more promising approach to let the domain experts work within domain-specific environments than educating software engineers to become experts in a variety of different disciplines. The role of the software engineer in the future should be to build domain-oriented environments in an interdisciplinary effort together with domain experts.

3. Knowledge-Based Design Environments

In this section an evolving framework and a number of system building efforts are discussed which try to instantiate our ideas about creativity enhancing environments.

3.1 Human Problem-Domain Communication and Construction Kits

To do anything in this world, it is not good enough to have weak methods, that is, to be smart in selective search and means-ends analysis [Simon 86]. In addition to that, we have to have knowledge about the task domain. Designers (e.g., architects, composers, user interface designers, database experts, knowledge engineers, etc.) are experts in their problem domain. The computer, because of its generality, can be a support tool for *all* knowledge workers. But domain specialists are not interested in learning the "languages of the computer"; they simply want to use the computer to solve problems and accomplish tasks. To shape the computer into a truly usable and useful medium, we have to make low-level primitives invisible and let users work directly on *their* problems and tasks. 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* [Fischer, Lemke 88] provides a new level of quality in human-computer communication by building important abstract operations and objects of a given area directly into the computing environment. In these systems, the designer can operate with personally meaningful abstractions, and the cognitive transformation distance between problem-oriented and system-oriented descriptions is reduced.

Many current knowledge-based systems use knowledge representations at a too low level of abstraction. This makes both system design and explanation difficult, since system designers have to transform the problem into a low-level implementation language and explanations require translating back to the problem level.

Construction kits are tools that foster 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. A construction kit makes the elements of a domain-oriented substrate readily available by displaying them in a menu or graphical palette (see Figure 3-1). This kind of system eliminates the need for prerequisite, low-level skills such as knowing the names of software components and the formal syntax for combining them.

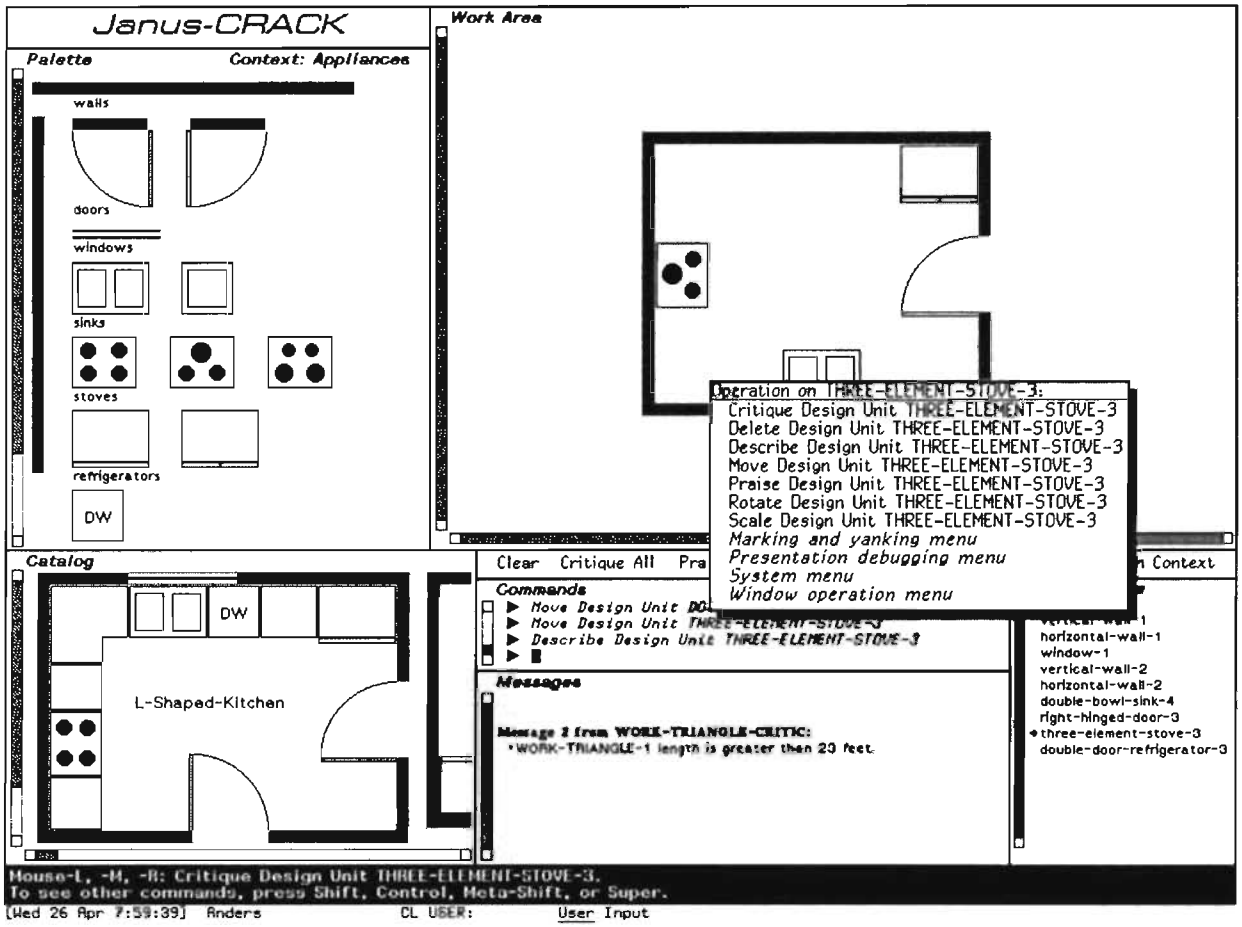

Figure 3-1: JANUS Construction Interface

JANUS is a knowledge-based system to support kitchen designers. The interface of JANUS's construction component is based on the world model [Hutchins, Hollan, Norman 86]. Design units are selected from the Palette, and moved into the work area. Operations on design units are available through menus. The screen image shown displays a message from the WORK-TRIANGLE-CRITIC.

Human problem-domain communication allows us to redraw the borderline between the amount of knowledge coming from computer systems versus coming from the application domain. Systems built to support human problem-domain communication establish an "application-oriented" vocabulary which is essential for effective reasoning and communication. They eliminate some of the opaqueness of computational systems by restricting them to specific application domains.

Construction Kits are not good enough. Evaluating the Pinball Construction Kit as a prototypical example against our objective to support human problem-domain communication, we have identified some shortcomings. The system helps to construct *an* artifact quickly, but it does not assist the designer in constructing *interesting and useful* artifacts. The pinball construction kit allows designers to build pinball machines in which balls get stuck in corners and in which devices may not be reachable. To assist designers in constructing truly interesting objects, design environments are needed.

3.2 Design Environments

Design environments give support that is not offered by simple construction kits. In addition to presenting the designer with the available parts and the operations to put them together, they incorporate knowledge about which components fit together and how they do so, and they contain cooperative critics that recognize suboptimal design choices and inefficient or useless structures. They support multiple specification techniques in creating a new system and understanding an existing one. They link internal objects and the external behavior and appearance, they provide animated examples and guided tours—techniques supporting the incremental development of a system. Following, two design environments will be described that we have constructed over the last few years:

- JANUS [Fischer, McCall, Morch 89a; McCall, Fischer, Morch 89; Fischer, McCall, Morch 89b] which supports architectural design and
- FRAMER [Lemke 89; Lemke, Fischer 90] which supports window-based user interface design.

JANUS: A Cooperative System for Kitchen Design

JANUS allows designers to construct artifacts in the domain of architectural design and at the same time informs them about principles of design and their underlying rationale by integrating two design activities: construction and argumentation. *Construction* is supported by a knowledge-based graphical design environment (see Figure 3-1) and *argumentation* is supported by a hypertext system (see Figure 3-2).

JANUS provides a set of domain-specific building blocks and has knowledge about properties and constraints of useful designs. With this knowledge it "looks over the shoulder" of users carrying out a specific design. If it discovers a shortcoming in the users' designs, it provides a critique, suggestions, and explanations, and assists users in improving their designs. JANUS is not an expert system that dominates the process by generating new designs from high-level goals or resolving design conflicts automatically. Designers control the behavior of the system at all times (e.g., the critiquing can be "turned on and off"), and if they disagree with JANUS, they can modify its knowledge base.

Critics [Fischer, Mastaglio 89] in JANUS are procedures for detecting nonsatisficing partial designs. JANUS' concept for integrating the constructive and argumentative component originated from the observation that a critique is a limited type of argumentation. The construction actions can be seen as attempts to resolve design issues. For example, when a designer is positioning the sink in the kitchen, the issue being resolved is "Where should the sink be located"?

The knowledge-based critiquing mechanism in JANUS bridges the gap between construction and argumentation. This means that critiquing and argumentation can be coupled by using JANUS' critics to provide the designer with immediate entry into the place in the hypertext network containing the argumentation relevant to the current construction task. Such a combined system provides argumentative infor-

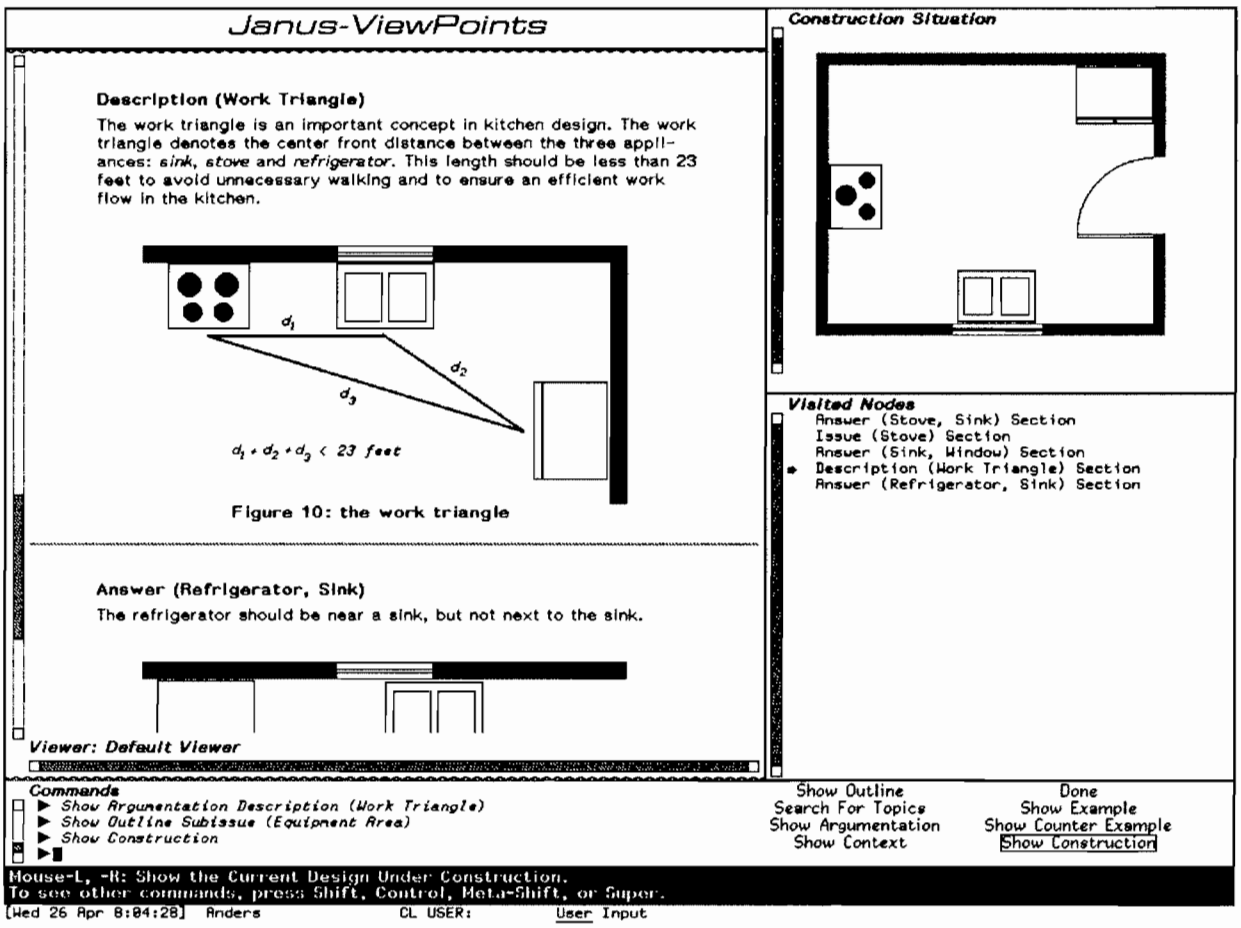

Figure 3-2: JANUS argumentation interface

JANUS's argumentation component uses the Symbolics Document Examiner as a delivery interface. The construction situation can be displayed in one of the panes to allow users to inspect the constructive and argumentative context simultaneously.

mation for construction effectively and efficiently, and designers do not have to realize before hand that information will be required, anticipate what information is in the system, or know how to retrieve it.

JANUS' Construction Component. The constructive part of JANUS supports building an artifact either *from scratch* or by *modifying an existing design.* To construct from scratch, the designer chooses building blocks from a design units PALETTE and positions them in the WORK-AREA (see Figure 3-1).

To construct by modifying an existing design, the designer uses the CATALOG (lower left in Figure 3-1), which contains several example designs. The designer can browse through this catalog of examples until an interesting one is found. This design can then be selected and brought into the WORK-AREA, where it can be modified.

The CATALOG contains both good designs and poor designs. The former satisfy all the rules of kitchen design and will not generate a critique. People, who want to design without having to bother with knowing

the underlying principles, might want to select one of these, since minor modifications of them will probably result in few or no suggestions from the critics. The *poor designs* in the CATALOG support learning the design principles. By bringing these into the WORK-AREA, users can subject them to critiquing and thereby illustrate those principles of kitchen design that are known to the system.

The *good designs* in the CATALOG can also be used to learn design principles and explore their argumentative background. This can be done by bringing them into the WORK-AREA then using the "Praise all" command. This command causes the system to generate positive feedback by displaying messages from all of the rules that the selected example satisfies. The messages also provide entry points into the hypertext argumentation.

JANUS' Argumentation Component. The hypertext component of Janus is implemented using Symbolics Concordia and Document Examiner software. The issue base is implemented using Concordia, a hypertext editor [Walker 88]. The Document Examiner [Walker 87] provides functionality for on-line presentation and browsing of the issue base by users.

When users enter the argumentative part of JANUS, they are brought into a section of the issue base relevant to their current construction situation. Their point of entry into the hypertext network contains the information required to understand the issue of interest. But argumentation on an issue can be large and complex so they can use this initial display of relevant information as a starting place for a navigational journey through the issue base, following links that will lead them to additional information. After examining the argumentative information, the designer can return to construction and complete the current task.

Critics as Hypertext Activation Agents. JANUS' knowledge-based critics serve as the mechanism to link construction with argumentation. They "watch over the shoulders" of designers, displaying their critique in the MESSAGES pane (center bottom in Figure 3-1) when design principles are violated. In doing so they also identify the argumentative context that is appropriate to the current construction situation.

For example, when a designer has designed the kitchen shown in Figure 3-1, the "Work-Triangle-Critic" fires and detects that the work triangle is too large. To see the arguments surrounding this issue, the designer has only to click on the text of this criticism with the mouse. The argumentative context shown in Figure 3-2 is then displayed.

FRAMER: A Design Environment for Window-Based User Interfaces

FRAMER [Lemke 89] is a design environment for window-based user interfaces. The overall design support provided by FRAMER is similar to JANUS. The system supports software design in multiple ways: making components readily available in a direct-manipulation interface, taking care of interdependencies of design characteristics by knowing about necessary design issues, and assessing designs by highlighting good aspects and suggesting improvements to eliminate the shortcomings of poor designs.

FRAMER promotes reuse and redesign [Fischer 87b]: the first step in the design is the selection of a framework from the library of reusable designs. Design with FRAMER always involves reuse. Figure 3-3 shows an intermediate design state. The system's knowledge about the design task is represented as a check list in the top left window. The check list describes the steps that a designer has to go through in designing a functioning interface. The "What you can do" window lists all design options related to the currently selected check list item. For each option the possible choices and an explanation of the sig-

nificance of the options is given. The descriptive aspects of the design are specified directly in this window, which functions as a dynamic, electronic form.

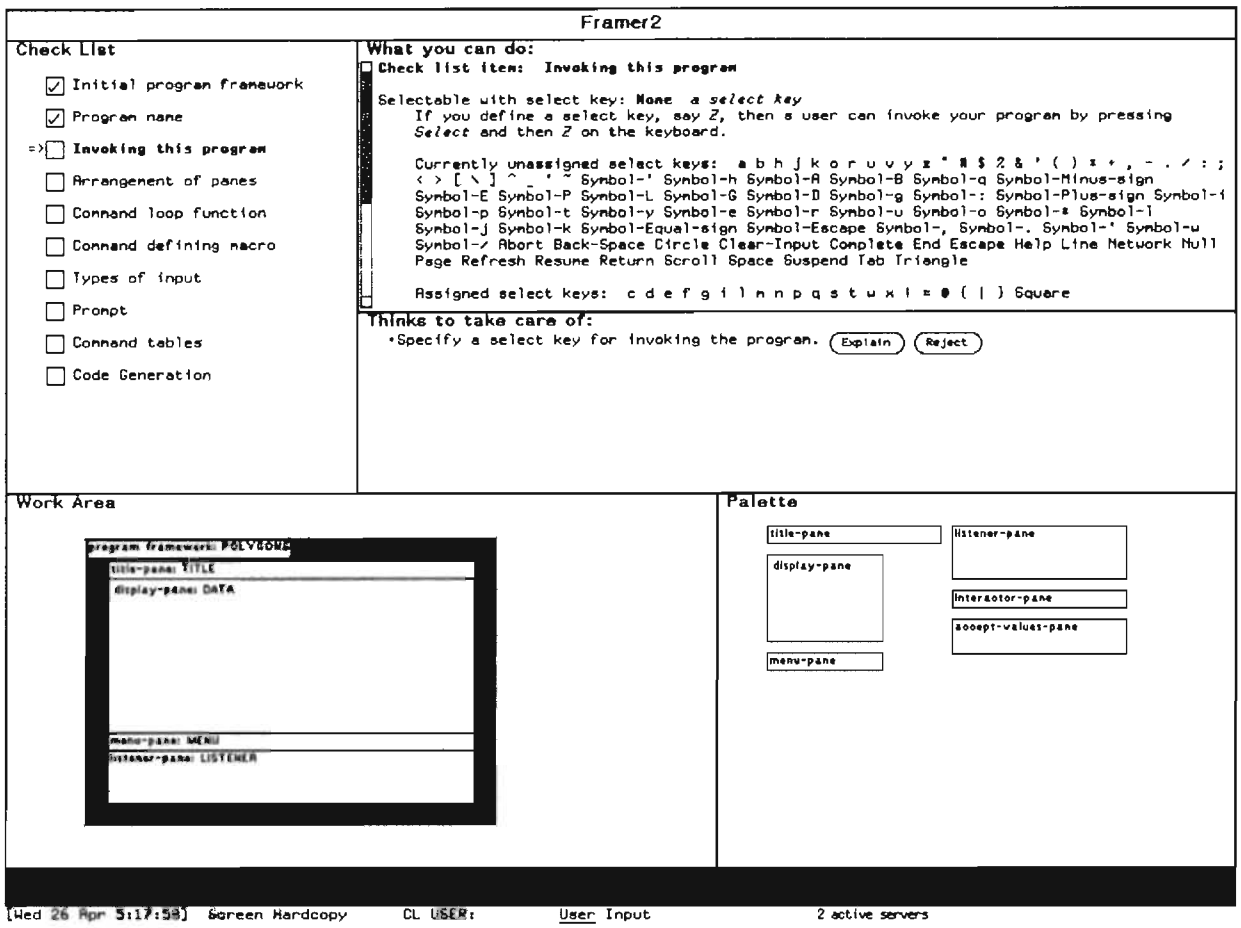

Figure 3-3: FRAMER—A Design Environment for Window-Based User Interfaces

The check list item "invoking the program" is currently selected. The "What you can do" window shows the issue what the select key (short cut) for invoking the program should be. The critic window shows a message recommending that a select key be defined. The designer can find out why this is recommended by pressing the "Explain" button, and then reject the critic if desired.

The "Things to take care of" window, below, continuously displays the related messages from the critics. Critics are classified as either mandatory or recommended. Mandatory critics describe conditions necessary for the interface to function, recommended critics suggest typical decisions according to criteria such as consistency and usefulness. Each type of critic message is associated with a prestored WHY and HOW explanation. Recommended critics can be overruled by clicking the "Reject" button. For some critic suggestions, the system knows a way of fixing up the problem. If so, an execute button appears near the suggestion ("Remedy" feature).

The work area contains a representation of the visual aspects of the design. It displays the window layout with the types and names of the windows. The window layout is generated and modified in a direct manipulation style using primitive elements from the palette on the right.

When all items in the check list are completed, the corresponding code can be generated. FRAMER can translate both ways between the representation used in FRAMER and executable code.

Experiences with FRAMER. The framework supported by FRAMER makes possible a major shift in the design process. In traditional environments the emphasis is on writing algorithms, whereas with high functionality computer systems supported by knowledge-based design environments, the emphasis is on *finding, understanding, and applying existing tools.* Designing applications in these environments means that few algorithms are created and that the descriptive part of software design grows correspondingly large.

FRAMER has turned out to be a powerful tool for rapid prototyping, which is of crucial importance for an ill-defined domain such as user interface design. FRAMER supports the co-evolution of specification and implementation in a natural way. The critics, the explanation component and the remedy feature allow the designers to use the environment to get some feedback about the quality of their designs.

As an artifact, FRAMER transforms a task by replacing a specification in terms of implementation concepts with an environment based on the "world model" metaphor [Hutchins, Hollan, Norman 86]. It changes the cognitive requirements for bridging the gap between the designer's intentions and actions on the system's interface. The direct manipulation metaphor for the graphical layout reduces the transformation distance from conceptual to executable representation. The continuous visualization allows the designer to directly monitor the characteristics of the artifact being created. With a system based on this metaphor, the designer can operate with a representation much more closely related to the designed artifact. FRAMER thus brings the task of designing a user interface much closer to the conceptual level at which the human interface designer operates.

3.3 End-User Modifiability

To enhance creativity in knowledge-based design environments, end-user modifiability [Fischer, Girgensohn 90] is of crucial importance, because these systems (instead of serving as general purpose programming environments) provide support for specific tasks. In cases where designers of these environments did not anticipate specific activities, users must be able to modify the design environment *itself.* There must be enough support structures to extend the environment at the conceptual level of the task domain—requiring only to descend as few layers as possible in the layered architecture (see Figure 4-1).

In JANUS, situations arise in which users want to design a kitchen with appliances that are not provided by the design environment (e.g., a microwave). Property sheets (see Figure 3-4) help users define new design unit classes or modify existing ones by eliminating the need to remember names of attributes. The modification process is supported with context-sensitive help (e.g., showing users constraints for the value of a field). The system supports the finding of an appropriate place for the new class in the class hierarchy by displaying the current hierarchy. Users can display the definition of every class in the hierarchy with a mouse click. In addition, the applicable critic rules for a class can be listed and the definition of each of these rules can be be displayed.

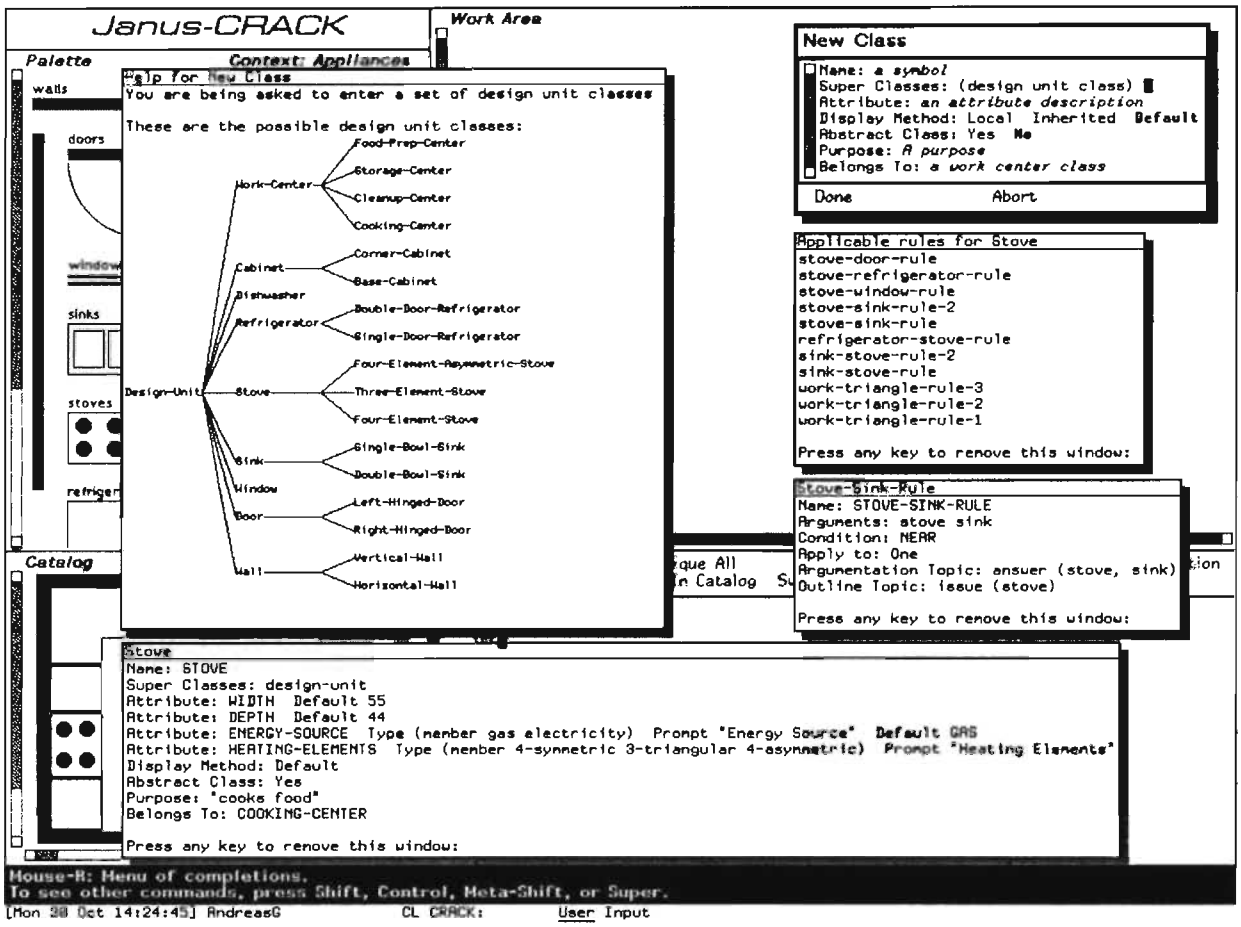

Figure 3-4: End-User Modifiability in JANUS: Introducing a Microwave

Users are supported in the modification of the design environment itself in a variety of ways: the property sheet with the title "New Class" guides the definition of a new class, the property sheet "Stove" shows the definition of a related class which can be modified, the "help" window visualizes the inheritance hierarchy of the objects known to JANUS, the "Applicable Rules for Stove" window shows the set of rules defining the critiquing knowledge for stove, and the "Stove-Sink-Rule" window shows a specific rule in detail.

4. Evaluation of our System Building Efforts

In this section I briefly describe what we have learned from designing and evaluating our knowledge-based design environments (for more details see [Lemke 89]).

The contribution of different system components. The contribution of the individual components of JANUS and FRAMER can be characterized as follows:

- *Palette:* Without the palette (i.e., starting with a general purpose programming language) the probability that a designer will construct a good kitchen, a good user interface or a good

pinball machine is basically zero. The domain-oriented abstractions contained in the palette enable the designer to think about the problems at the right level of abstraction.

- *Catalog:* Developing a rich knowledge base and supporting reuse and redesign were discussed as important enabling conditions for creativity in section 2.2. These are the functions served by the Catalog. It characterizes the space of possible designs and gives designers a feel for what kinds of artifacts can be constructed. In the current systems the catalog consists only of a set of drawings—the system has no "understanding" of the objects contained in the catalog. This lack of knowledge implies that an intelligent information retrieval system [Fischer, Nieper-Lemke 89] cannot be constructed to assist designers in finding the design which comes closest to their goals.
- *Interaction style:* The direct manipulation style [Hutchins, Hollan, Norman 86] supported by the design environments allows designers to operate in "world model". They can deal with the objects in the way that they think about them, thereby avoiding the need for manipulating textual descriptions of the objects. Carrying out the construction activities within the JANUS system allows the system to become aware of what the human is doing without requiring additional elaborate procedures to map an external world into the world of the system.
- *Checklist:* The checklist in FRAMER (see Figure 3-3) provides some guidance for designers in carrying out a task. Components of this kind should remain optional, reminding designers of the things they should consider, without imposing a specific way of doing things.
- *Critics:* Critics are a major step for extending construction kits to design environments. They provide the important knowledge-based component that can analyze a specific design and provide feedback (criticism and praise) to the designer. For non-trivial design tasks, they are of crucial importance.
- *Explanation:* Comments and advice provided by the critics are not necessarily understood and an explanation component provides a rationale for the critic's world view.
- *Argumentation:* In design there are no best solutions. Therefore, design environments must acknowledge and support the judgmental nature of design. The goals of our design environments are to inform the designer and to evoke alternative understandings, not authoritarian judgement and decision making.

Further empirical research is needed to determine the precise conditions (with respect to the goals and knowledge of individual designers) under which these environments enhance or hinder creativity.

Let the situation talk back. Design environments enhance prototyping and allow the "situation to talk back" [Schoen 83], supporting the incremental construction of models. The critics give designers an opportunity to learn on demand. The argumentative component makes models open, discussable and defendable. The problem domain level of the interaction illustrates the important relationships and supports not only finding a solution, but allows the designers to convince themselves with less effort that the solution is what they had in mind, to retain solutions with less cognitive effort and to explain it and discuss it with others.

Breakdowns as a source for creativity. The critics indicate to designers that they have violated a design principle. This "breakdown" in construction provides the "knowledge in the world" which serves as the activation pattern for "knowledge in the head" and in the argumentation part of the design environment. Without an integrated system, this synergy cannot be exploited. In many other system building efforts, the development of computer support for construction and argumentation has proceeded in parallel with little or no interaction between them. The former is associated with computer-aided design (CAD), the latter with hypertext—in particular what is known as IBIS hypertext [Conklin 87]. The major advantage offered by this integration is that designers can work towards the construction of an intelligible entity

rather than on the acquisition of knowledge and facts without a context in which they can be immediately used and understood. The integrated design environment presents relevant argumentation when it still makes a difference to the decision taken on the issue. This architecture supports "reflection-in-action" [Schoen 83; McCall, Fischer, Morch 89] in which the designer acts and the situation talks back. Without such an environment, designers may not see the unintended consequences of their constructive actions. Schoen points out that such unintended consequences—pleasant or unpleasant—are the crucial stimulus to reflection. Many problem situations do not speak for themselves, they need a spokesman such as a critic.

Layered Architectures. *Human problem-domain communication* (see Section 3.1) turns the computer into an invisible instrument allowing knowledgeable, task-oriented scientists and designers to work with the abstractions and concepts of their domains. To achieve this goal, environments with an underlying layered architecture are needed. Figure 4-1 shows the layered architecture underlying the JANUS system.

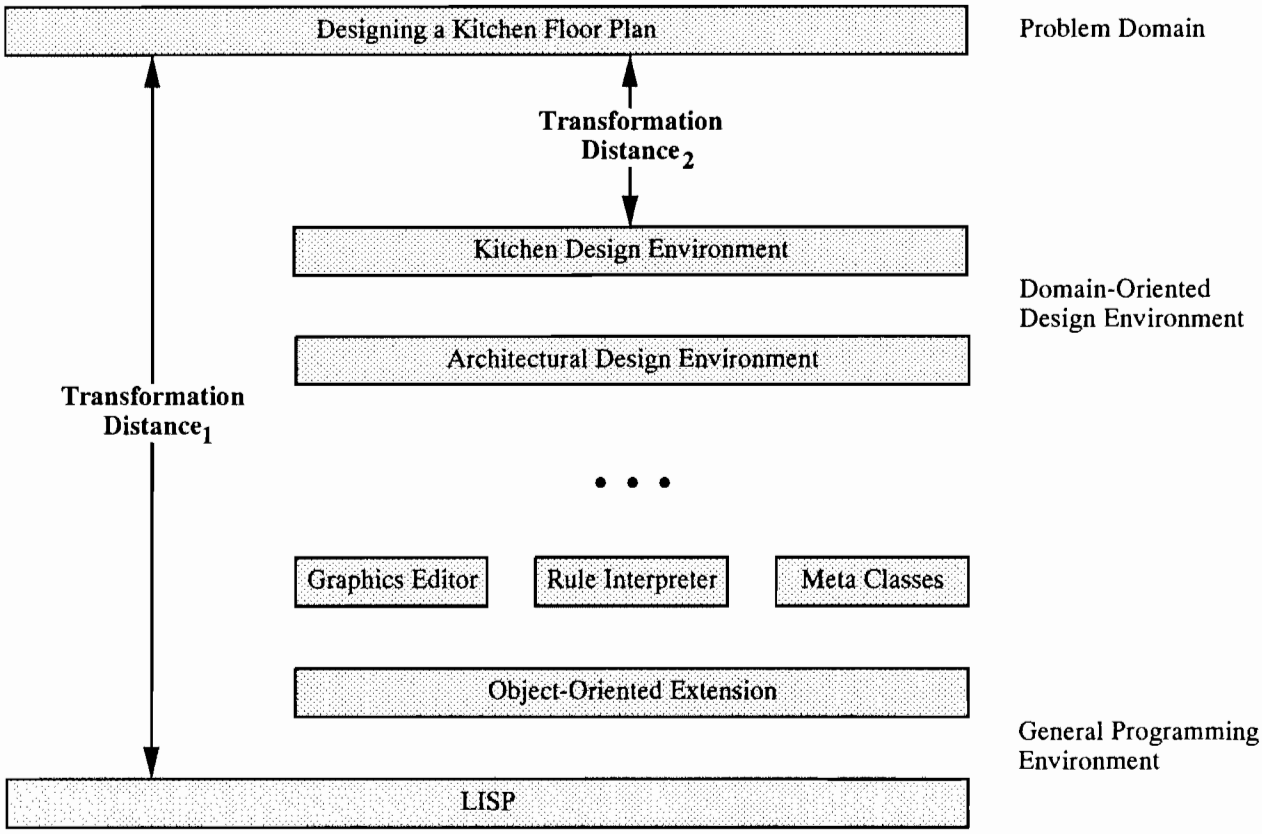

Figure 4-1: Layers of Abstraction to Reduce the Transformation Distance between Problem Domain and System Space

Transformation distance$_1$, resulting from an implementation based on a programming language, is considerable larger than transformation distance$_2$, which occurs by building a system with a design environment.

Layered architectures support end-user modifiability, because users can change the behavior of the sys-

tem in the layers near the top allowing them to remain in the context of the problem space. If a change extends beyond the functionality provided by any given layer, users are not immediately thrown back to the system space but can descend one layer at a time.

Augment the skill of designers—do not "de-skill" them. Our design environments are prototypical instances of a class of systems called *cooperative problem solving systems* [Fischer 90; Fischer et al. 90]. These systems are fundamentally different from traditional expert systems (such as MYCIN [Buchanan, Shortliffe 84]). Expert systems automate designers' tasks and confront them with solutions, whereas cooperative problem solving systems augment the designers' capabilities and negotiate solutions with them. Systems such as JANUS and FRAMER aim to inform and support the judgment of designers, they do not "de-skill" them by judging or designing for them [Bodker et al. 88]. The designer working with these systems is free to ignore, turn off, or alter the criticism displayed.

Cooperation requires more from a system than having a nice user interface or supporting natural language dialogs. One needs a richer theory of problem solving, which analyzes the functions of shared representations (i.e. models of the communication partner, models of the task), mixed-initiative dialogues, argumentation and management of trouble. In a cooperative problem solving system, the designer and the system share the problem solving and decision making and different role distributions may be chosen depending on the user's knowledge, the user's goals and the task domain. A cooperative system requires much richer communication facilities than the ones that were offered by traditional expert systems. Any cooperative system raises important questions regarding (a) which part of the responsibility has to be exercised by human beings and which part by the systems and (b) how should things be organized so that the intelligent part of the automatic system can communicate effectively with the human part of the intelligent system.

5. Conclusions

Creativity enhancing environments need to be different from traditional computer systems. Architectures must be developed that will put the domain experts into the driver's seat and that will allow them to make major contributions to the development of domain-specific design environments [Gero 89]. Many intellectual disciplines (such as architectural design, programming, composing, writing, etc.) should not remain primarily "spectator sports" done by a few and observed by many. Convivial tools [Illich 73] are needed to break down the boundary between programming and using programs. Our design environments are a step into this direction. We are convinced that the texture of the computer tools of the future will depend upon the people who design and use them. To the extent that domain experts from different disciplines (e.g., artistic, musical and literary people) will find it worthwhile to become computer-literate and make use of the new medium, the medium itself will reflect the wide range of human experience. Computer systems will lack those qualities if we isolate such people from computers.

Acknowledgements. The author would like to thank Andreas Lemke, who developed the FRAMER system, Anders Morch, who developed the JANUS system, Andreas Girgensohn, who extended JANUS to be modifiable by end users, and Raymond McCall, who provided insights in architectural design issues and their support through hypertext systems. Hal Eden and Christian Rathke provided valuable comments on an earlier draft of the paper.

The research was partially supported by grant No. IRI-8722792 from the National Science Foundation, grant No. MDA903-86-C0143 from the Army Research Institute, grants from the Intelligent Systems Group at NYNEX and from Software Research Associates (SRA), Tokyo and support from Artur Fischer.

References

[Bentley 86]
J.L. Bentley, *Document Design*, Communications of the ACM, Vol. 29, No. 9, September 1986, pp. 832-839.

[Bodker et al. 88]
S. Bodker, J.L. Knudsen, M. Kyng, P. Ehn, K.H. Madsen, *Computer Support for Cooperative Design*, Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88), ACM, New York, September 1988, pp. 377-394.

[Buchanan, Shortliffe 84]
B.G. Buchanan, E.H. Shortliffe, *Human Engineering of Medical Expert Systems*, in B.G. Buchanan, E.H. Shortliffe (eds.), *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley Publishing Company, Reading, MA, 1984, pp. 599-612, ch. 32.

[Cherry 81]
Lorinda Cherry, *Computer Aids for Writers*, Proceedings of the ACM SIGPLAN SIGOA Symposion on Text Manipulation, Portland, Oregon, 1981, pp. 61-67.

[Conklin 87]
J. Conklin, *Hypertext: An Introduction and Survey*, IEEE Computer, Vol. 20, No. 9, September 1987, pp. 17-41.

[Curtis, Krasner, Iscoe 88]
B. Curtis, H. Krasner, N. Iscoe, *A Field Study of the Software Design Process for Large Systems*, CACM, Vol. 31, No. 11, November 1988, pp. 1268-1287.

[diSessa, Abelson 86]
A. diSessa, H. Abelson, *Boxer: A Reconstructible Computational Medium*, Communications of the ACM, Vol. 29, No. 9, September 1986, pp. 859-868.

[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 90]
G. Fischer, *Communication Requirements for Cooperative Problem Solving Systems*, The International Journal of Information Systems (Special Issue on Knowledge Engineering), Vol. 15, No. 1, 1990, pp. 21-36.

[Fischer et al. 90]
G. Fischer, A.C. Lemke, T. Mastaglio, A. Morch, *Using Critics to Empower Users*, Human Factors in Computing Systems, CHI'90 Conference Proceedings (Seattle, WA), ACM, New York, April 1990, pp. 337-347.

[Fischer, Girgensohn 90]
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, pp. 183-191.

[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, pp. 179-222.

[Fischer, Mastaglio 89]
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.

[Fischer, McCall, Morch 89a]
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.

[Fischer, McCall, Morch 89b]
G. Fischer, R. McCall, A. Morch, *JANUS: Integrating Hypertext with a Knowledge-Based Design Environment*, Proceedings of Hypertext'89, ACM, New York, November 1989, pp. 105-117.

[Fischer, Nieper-Lemke 89]
G. Fischer, H. Nieper-Lemke, *HELGON: Extending the Retrieval by Reformulation Paradigm*, Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX), ACM, New York, May 1989, pp. 357-362.

[Gero 89]
J.S. Gero, *A Locus for Knowledge-Based Systems in CAAD Education*, The CAAD Futures '89 Conference, Havard University, Cambridge, June 1989, Pre-Publication Edition.

[Hayes 78]
J.R. Hayes, *Cognitive Psychology - Thinking and Creating*, Dorsey Press, Homewood, Illinois, 1978.

[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.

[Jacob 77]
Jacob, *Evolution and Tinkering*, Science, June 1977, pp. 1161-1166.

[Kaplan,Simon 89]
C.A. Kaplan, H.A. Simon, *In Search of Insight*, Technical Report, Carnegie-Mellon University, May 1989.

[Lemke 89]
A.C. Lemke, *Design Environments for High-Functionality Computer Systems*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, July 1989, Publication No. 9009170, University Microfilms Inc., Ann Arbor, MI.

[Lemke, Fischer 90]
A.C. Lemke, G. Fischer, *A Cooperative Problem Solving System for User Interface Design*, Proceedings of AAAI-90, Ninth National Conference on Artificial Intelligence, 1990, pp. 479-484.

[McCall, Fischer, Morch 89]
R. McCall, G. Fischer, A. Morch, *Supporting Reflection-in-Action in the Janus Design Environment*, Proceedings of the CAAD Futures '89 Conference, Harvard University, Cambridge, June 1989, Pre-Publication Edition.

[Newell, Simon 75]
A. Newell, H.A. Simon, *Computer Science as Empirical Inquiry: Symbols and Search*, Communications of the ACM, Vol. 19, No. 3, March 1975, pp. 113-126.

[Norman 88]
D.A. Norman, *The Psychology of Everyday Things*, Basic Books, New York, 1988.

[Papert 86]
S. Papert, *Constructionism: A New Opportunity for Elementary Science Education*, Proposal to The National Science Foundation, MIT - The Media Laboratory, Cambridge, MA, 1986.

[Rittel 72]
H.W.J. Rittel, *On the Planning Crisis: Systems Analysis of the First and Second Generations*, Bedriftsokonomen, No. 8, 1972, pp. 390-396.

[Schoen 83]
D.A. Schoen, *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York, 1983.

[Simon 73]
H.A. Simon, *The Structure of Ill-Structured Problems*, Artificial Intelligence, No. 4, 1973.

[Simon 81]
H.A. Simon, *The Sciences of the Artificial*, The MIT Press, Cambridge, MA, 1981.

[Simon 86]
H.A. Simon, *Whether Software Engineering Needs to Be Artificially Intelligent*, IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, July 1986, pp. 726-732.

[Walker 87]
J.H. Walker, *Document Examiner: Delivery Interface for Hypertext Documents*, Hypertext'87 Papers, University of North Carolina, Chapel Hill, NC, November 1987, pp. 307-323.

[Walker 88]
J.H. Walker, *Supporting Document Development with Concordia*, IEEE Computer, Vol. 21, No. 1, January 1988, pp. 48-59.