A Tool for Achieving Consensus in Knowledge Representation[1]

Loren G. Terveen

AT&T Bell Laboratories
600 Mountain View Avenue
Murray Hill, NJ 08827
terveen@research.att.com

David A. Wroblewski

U S West Advanced Technologies
4001 Discovery Drive
Boulder, CO 80303
davew@uswest.com

Abstract

We develop the notion that knowledge editing is a cooperative activity that requires knowledge editors to reach *consensus* as they represent information in a knowledge base. We describe an intelligent knowledge editing tool, the *HITS Knowledge Editor*, and illustrate how it assists knowledge editors in reaching consensus.

Introduction

For the past several years, we have been studying the task of *knowledge editing* and have constructed a tool – the *HITS Knowledge Editor (HKE)* – based on our understanding of the task. A theme that has emerged is that knowledge editing is a cooperative activity that requires knowledge editors to reach *consensus* as they represent information in a knowledge base (Hill 1989). In this paper, we describe the role of consensus in knowledge editing, discuss how HKE supports users in achieving consensus, and illustrate the process of reaching consensus with examples taken from user studies.

The Activity of Knowledge Editing

Knowledge editing involves the entry, viewing, access, and maintenance of information in a knowledge base. Knowledge editing is difficult for many reasons; we state three that are relevant to this paper. First, users often modify an existing knowledge base rather than starting from scratch; therefore, the representational decisions they make must be in harmony with existing information and representational conventions. Second, there is no single correct representation of a domain; therefore, any representational design involves reasoned identification, deliberation, and resolution of representational issues. Third, knowledge bases often are designed and built by groups of knowledge editors. Since there is no right solution, representational decisions must be understood by all members of a design group.

The HITS Knowledge Editor (HKE) has been designed to assist users in the task of knowledge editing. HKE is an interface to CYC (Lenat & Guha, 1990). Two of its design features are relevant to this paper. First, it provides a *workspace* – a sort of sketchpad – for user-system problem solving. This lets users sketch graphs representing new information they intend to enter into the knowledge base. When users are satisfied with a sketch, they request HKE to incorporate it into the knowledge base. The second relevant feature of HKE is that it includes *design critics* (Fischer, Lemke, Mastaglio & Morch 1990) that assist users in incorporating new information. Critics deliver various sorts of assistance (Terveen 1991); in this paper we discuss only *troubles* – inconsistencies between proposed information and the existing knowledge base.

HKE embodies an analysis of knowledge editing into six sub-activities (Terveen 1991). Briefly, these sub-activities are defined as follows:

- During *exploration*, users view the knowledge base in order to understand its contents and the representational conventions that guided its construction.
- During *aggregation*, users gather objects from the knowledge base that are potentially relevant for the work at hand and place them in the sketch.
- During *specification*, users sketch out new knowledge they intend to enter into the knowledge base. A sketch is a buffer between users and the knowledge base; the KB is not modified until the *incorporation* activity.
- During *annotation*, the users or the system makes notes about the state of the work in progress, usually describing some outstanding issue.
- During *incorporation*, the system merges the specification into the knowledge base and computes assistance. Incorporation is done at the users' request, when they are satisfied with the specification.
- During *repair*, the system presents the issues it has detected and works with the users to resolve them.

This paper focuses on the activities of specification, incorporation, and repair, since they constitute knowledge entry, and it is primarily during the entry of new knowledge that the problem of reaching consensus arises.

Consensus in Knowledge Editing

One of the factors that makes building knowledge bases difficult is that there is no single correct representation of a domain. Representing a domain requires knowledge editors to identify and resolve design issues. The thesis of

[1]This work was done while both authors were at the MCC Human Interface Laboratory.

this paper is that this process of achieving *consensus* is a critical element of knowledge editing. We introduce the notions of *synchronous* and *asynchronous* consensus and illustrate them with examples taken from user studies done as a part of (Terveen 1991).

We define consensus as a mutual understanding of a design issue, how the issue was resolved, and the reasons why that resolution was chosen. This means that two designers have achieved consensus even if they disagree on the resolution, as long as either can use the rationale behind it in understanding the state of the knowledge base. For example, consider the (apparently) simple task of representing your family in a knowledge base. Suppose you first deal with the case of representing who is married to whom. A number of issues instantly arise. How should individuals be represented? Should there be a single class named `Person` (we denote objects in the knowledge base using `Courier` font), and what should the relevant subclasses be? `Male` and `Female`? Perhaps there should instead be a slot called `sex` that can hold an appropriate value? Should there be one reflexive relation `spouse` or two relations `husband` and `wife`, each asserted to be the inverse of the other? In either case, is the slot single or multiple valued? If it is single valued, is polygamy then impossible to represent? Does that matter for this knowledge base? What about divorced or separated couples? What about widows and widowers? This gives a flavor of the type of issues that must be resolved in even apparently simple domains. If a work group gathers together to resolve such issues, we refer to this process as reaching *synchronous* consensus.

Contrast this with the following situation. Again you wish to represent your family, but discover upon inspecting the knowledge base that an ontology for families already exists. Now you must evaluate the existing ontology, determine if it is suitable for your family, and modify it if necessary. Unfortunately, if you modify the knowledge base you may have to update all the families already represented under the current ontology. In order to perform this evaluation, you must inform yourself of the design decisions inherent in this ontology and the factors that influenced those decisions. You must, in other words, reach consensus with the original designers of that data; your modifications to the knowledge base must be consistent with the existing design. This is *asynchronous* consensus – multiple designers, separated by time and perhaps space, achieve a shared understanding of a domain mediated by a formal representation of that domain in the knowledge base.

Figure 1 illustrates the most important parameters in building consensus – spatial and temporal distance – and summarizes how HKE supports achieving consensus.

Designers working on the same problem at the same time must achieve synchronous consensus. HKE provides facilities that support designers who are working on the same problem at the same time and *in the same place*, i.e., in front of a workstation running HKE.

We believe that asynchronous consensus is of critical importance throughout the design, implementation, and evolution of a knowledge base. A knowledge base often is changed to meet new needs or to remedy newly discovered shortcomings, long after the original design has been in place, and long after the original designers have left the organization. Nevertheless, attempts to change the design must proceed from an understanding of the issues driving the existing design.

This topic has been the focus of several "design recovery" projects. Carroll's (1990) "claims extraction" can be viewed as a process by which the principles behind existing interfaces can be brought to light, even if they were not articulated by the original designers in any medium but the final artifact. Biggerstaff (1988) has looked at methods of "recovering" a design specification from a piece of software. Both of these efforts have focused on extracting the design issues from a finished work; our work differs in that we assume that the design process is never completely finished, but merely slowly evolving. Shared, evolving knowledge bases require new designers to comprehend the consensus underlying the existing design of the knowledge base; thus HKE attempts to mediate constructing and maintaining a consensus throughout this evolutionary process.

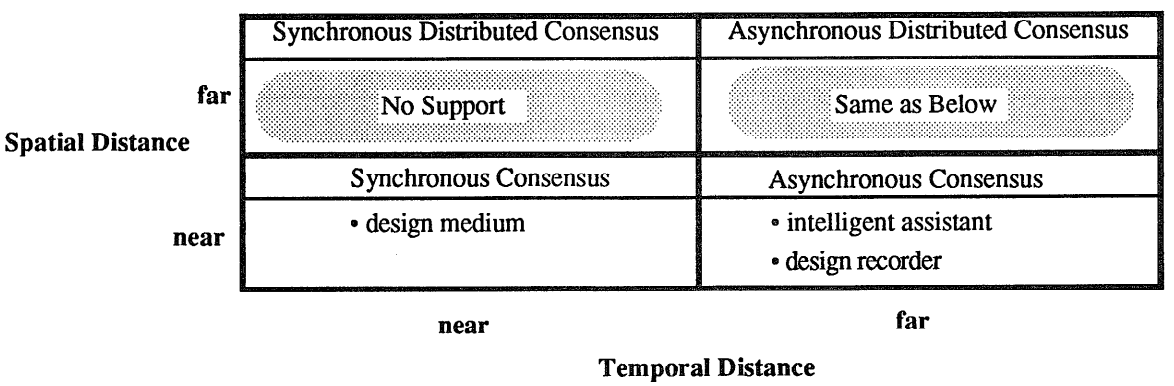


Figure 1: Dimensions of consensus and HKE facilities to support them.

The HITS Knowledge Editor (HKE)

HKE mediates the process of achieving consensus in three ways.

- It serves as a *design medium* – sketches allow users to surface and track representational issues that must be resolved as they reach synchronous consensus.
- It serves as an *intelligent assistant* – HKE raises issues during incorporation (these are symptoms of a lack of consensus with the existing design) and then works with users to resolve them during repair.
- It serves as a *design recorder* – sketches capture significant aspects of the design activity. Sketches can be stored and reused, making parts of the design process available as a resource to other users in the future.

We illustrate each of these properties with an episode taken from a user study (Terveen 1991). In the study, pairs of subjects were given the task of representing knowledge about the structure of their organization (the Artificial Intelligence or Human Interface Laboratory at MCC), including researchers and their areas of expertise, research projects, and software systems.

Example 1: Reaching Synchronous Consensus in a Design Medium

When a group of knowledge editors represent a domain, they must achieve synchronous consensus, i.e., they must identify and resolve representational issues. HKE supports this process with a direct manipulation interface that makes it easy for users to surface and track issues. Users specify new information by sketching a graph of objects and their relationships. The sketch is a buffer between the users and the knowledge base; the KB is not modified until the users are satisfied with their sketch. When they are, they request HKE to incorporate it into the KB, which it does automatically.

This example shows how HKE supported one pair of subjects (we'll refer to them as subjects 3 & 4) in resolving the representational issue "should researchers be represented as working for a research laboratory or a particular project within the laboratory?" Figure 2 shows an early point in their work. They have stated the super organization of the lab, its manager, several of the researchers, and one of its sub-projects.

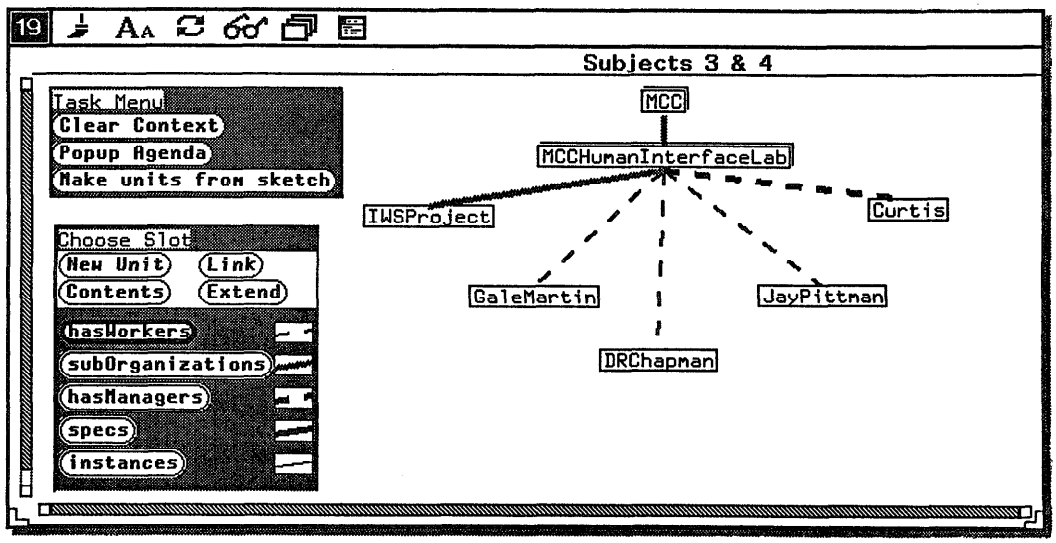
At this point, subject 3 raises a new issue:

> Maybe I should put those three (*using the mouse to indicate the objects* GaleMartin, DRChapman, *and* JayPittman) under that (*indicating the object* IWSProject), and break these links (*indicating the links from* MCCHumanInterfaceLab *to* GaleMartin, DRChapman, *and* JayPittman) ... or maybe they stay there.

Subject 4 responds:

> You can always restructure projects and you still work for the lab.

At this point, however, the subjects remain unsure about how to resolve this issue, so they leave the sketch as is and continue on in their task. About 10 minutes later, they have specified another of the sub-projects of the MCCHumanInterfaceLab, HITSProject, and several of the researchers of this project. The state of their work is shown in figure 3.



Double lines around an icon indicate that an object by this name exists in the knowledge base, e.g., MCC exists but Curtis does not. The Choose Slot menu functions like the legend of a map. Each slot has an associated line pattern, e.g., hasManagers is represented by a thick dashed line. The icons for two objects related by one of these slots are linked by the appropriate line pattern, e.g., MCCHumanInterfaceLab and Curtis are related by the hasManagers slot, so their icons are linked by a thick dashed line.

Figure 2: Surfacing a representational issue in a sketch

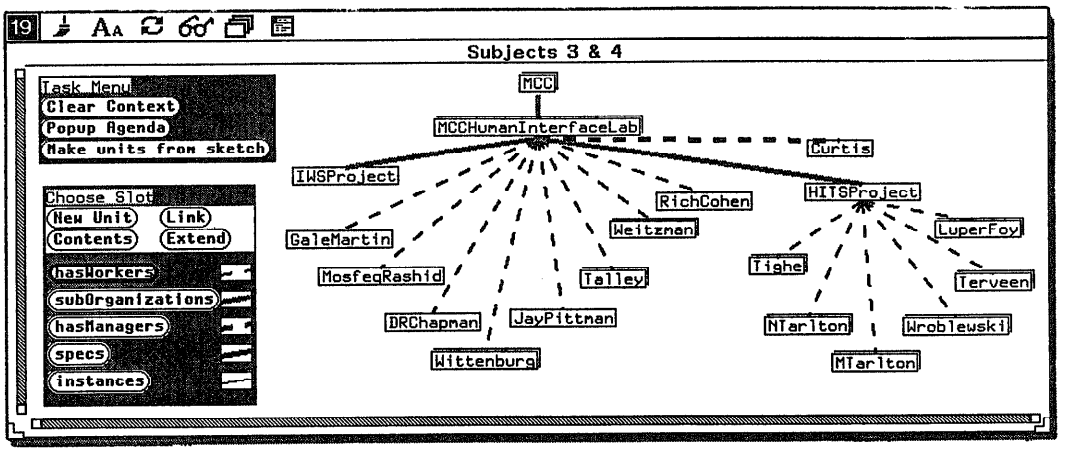


Figure 3: Noting an inconsistent resolution of a representational issue

At this point, it becomes apparent that they have not resolved consistently the issue "should researchers be represented as working for a research laboratory or a particular project within the laboratory?" Subject 4 says:

> You've now made the picture inconsistent.

Subject 3 responds:

> Yeah, I have. I should move these (*indicating the researchers linked to* MCCHumanInterfaceLab) over here (*indicating* IWSProject).

They then do so, thus resolving the issue by associating researchers with the most specific organization for which they work.

To summarize, sketches serve as an external memory that aids users in tracking and resolving issues. Terveen (1991) shows that other tools for CYC do not have this property, and this can lead users to lose track of issues. Notice that so far it is not even important that HKE is a computer-based tool – after all, even a whiteboard could help designers track issues. The next example shows additional advantages of HKE due to the fact that sketches are machine-interpretable.

Example 2: Reaching Asynchronous Consensus Assisted by an Intelligent Agent

A knowledge base embodies representational decisions that reflect a consensus view of a domain. Edits to the knowledge base must be based on an understanding of the existing consensus – lack of understanding manifests itself as trouble that occurs while trying to update the knowledge base. This section considers a particular representational decision embodied in CYC and shows how HKE helped subjects to detect an inconsistency between that decision and one they had made. The issue is "should a particular computer program (like HKE) be represented as a subclass or an instance of the general concept of a ComputerProgram?" In the knowledge base, particular programs are represented as subclasses of the class ComputerProgram; however, two subjects (we'll refer to them as subjects 7 & 8) decided to make particular programs instances of ComputerProgram.

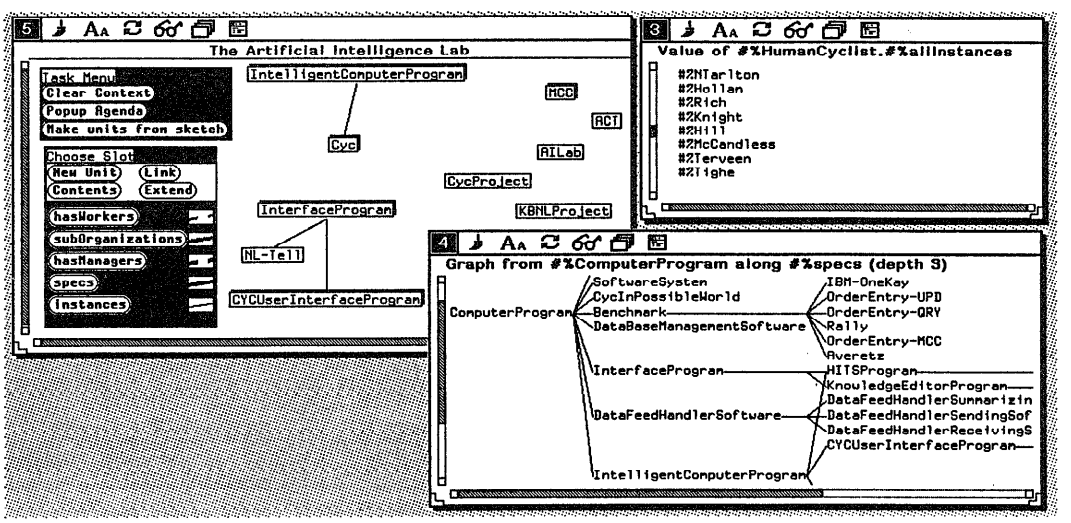


Figure 4 – Problem in achieving asynchronous consensus

Figure 4 shows an early point in the work of subjects 7 & 8 as they were representing the MCC AI Lab. Windows 3 and 4 show exploratory information-gathering moves into the existing knowledge base. Window 5 shows the sketch the subjects are constructing. Notice that the class hierarchy in window 4 shows that both CYCUserInterfaceProgram and InterfaceProgram already are represented as subclasses of ComputerProgram in the knowledge base. Through additional inheritance, both objects already are known to be derived instances of the class Collection, the set of all sets of things. However, in the subjects' sketch, they stated that CYCUserInterfaceProgram is an instance of InterfaceProgram, which through inheritance will make CYCUserInterfaceProgram an instance of ComputerProgram, and through additional inheritance, an instance of the class IndividualObject. However, no object can be an instance of both Collection and IndividualObject – they are declared to be *mutually disjoint.*

Recall that a sketch is a buffer, and that the knowledge base is not updated until the users request HKE to incorporate a sketch. When they do so, and HKE attempts to incorporate the assertion instanceOf(CYCUserInterfaceProgram, InterfaceProgram) into the knowledge base, it detects this trouble. Users then can access the repair resource shown in figure 5 as an aid in understanding and resolving the problem. This resource suggests that the trouble be repaired by making CYCUserInterfaceProgram a *subclass* (rather than an instance) of InterfaceProgram. This repair option is appropriate since CYCUserInterfaceProgram and InterfaceProgram both are members of the class hierarchy of computer programs.

To summarize, HKE detects a trouble when attempting to incorporate an assertion into the knowledge base. The trouble is a symptom of a lack of consensus between previous knowledge editors and the current knowledge editors. HKE suggests a repair action that may fulfill the intention of the current knowledge editors and does fit in with the existing consensus. The principle underlying both examples 1 and 2 is that HKE supports users in achieving consensus by making symptoms of non-consensus visible.

Example 3: Achieving Asynchronous Consensus through Design Recording

A sketch embodies aspects of users' problem solving activity. Sketches can be stored and used as a resource for future representational activity; thus, problem solving does not have to be duplicated. Figure 6 shows the completed sketch for subjects 7 & 8. This sketch embodies several decisions useful for similar tasks.

- It shows the *vocabulary* (classes and slots) that two users have found appropriate for representing a domain.
- It records particular facts that users asserted about objects in the domain. This information can be used as a template by users representing a similar domain.
- It filters out information about objects that users do not consider important.

This sketch could be useful to other users representing an organization. They would not have to do as much exploration of the existing knowledge base to find relevant information and filter out irrelevant information, they could use the vocabulary chosen by subjects 7 & 8, and they could represent much of their domain simply by copying and editing the sketch of subjects 7 & 8. Note that these benefits do not require extra work on the part of knowledge editors – they are not forced to go "off-task" to document their work.

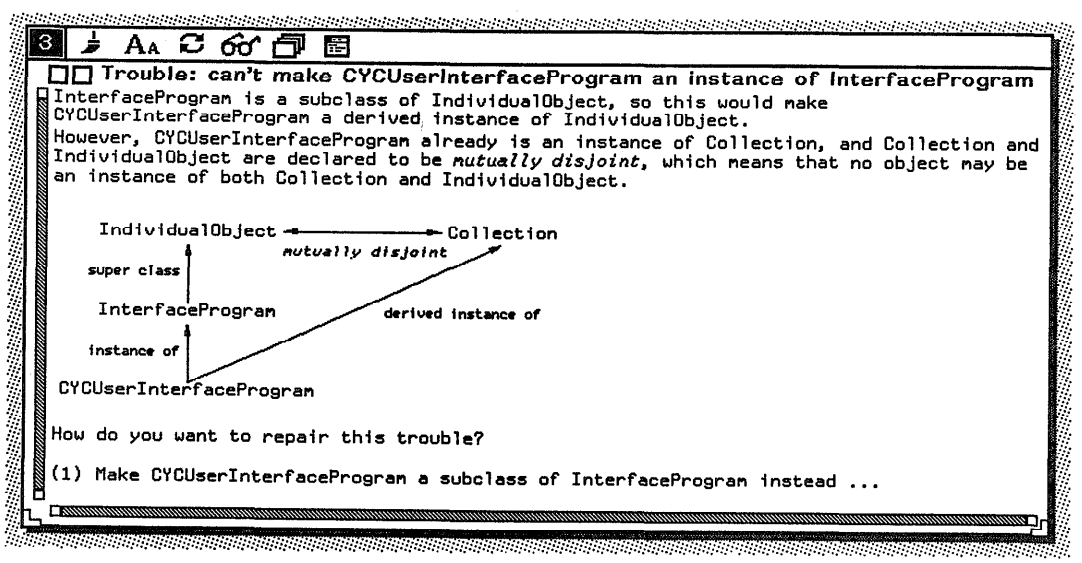


Figure 5 – A repair resource

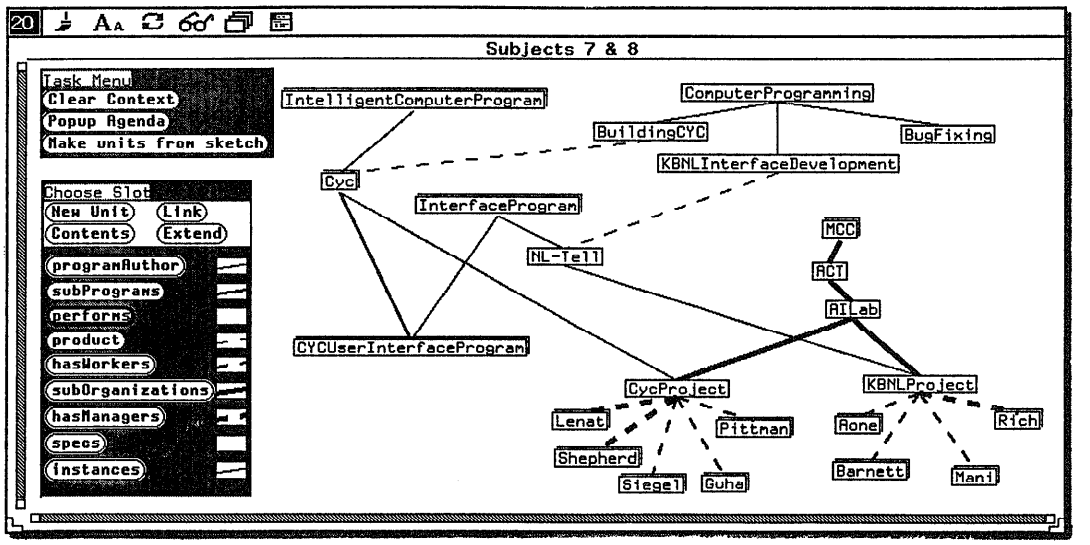


Figure 6 – A sketch as a resource for future activity

Summary: Strengths and Weaknesses

HKE supports users in achieving consensus in three ways. It provides a *design medium* that allows users to surface and track issues that must be resolved to reach synchronous consensus. It offers *intelligent assistance* that helps users work in harmony with representational decisions embodied in the knowledge base. And it serves as a *design recorder*, capturing significant aspects of the design activity that can be used as a resource for doing similar tasks in the future.

There are three ways that HKE does not support users in reaching consensus. First, it does not capture the rationale behind representational decisions. *Issue-based information systems* (Conklin & Begeman 1987, Fischer, McCall, & Morch 1989) are a promising approach to capturing design rationale. However, they require designers to go off-task to document their work, and software developers are notoriously loathe to do this sort of documentation.

Second, sketches do not capture repair activity. While HKE provides repair resources like the one shown in figure 5, it does not update the sketch to record either the repair decision or the reasoning behind it. We have experimented with special sketches that graphically depict a trouble and resources for solving it. This allows the trouble, the problem solving work done by users to resolve it, and the resolution to be stored and available as resources for subsequent representational activity.

Finally, HKE currently offers no support in achieving synchronous *distributed* consensus, i.e., to people who are geographically separated but are working on a shared knowledge base at the same time. We have experimented with methods that (1) allow users to mark sections of the knowledge base as being of personal interest, and (2) embed agents in the knowledge base that watch for other users to access these sections and notify the original user.

Acknowledgements

We thank Will Hill for identifying the issue of consensus, Steven Tighe for implementing HKE's sketching functionality, and Gerhard Fischer for his constructive comments on a version of this paper that we presented at the 1991 HCIC workshop.

References

Biggerstaff, T. (1988). Design Recovery for Maintenance and Reuse. TR STP-378-88. Austin Texas: MCC.

Carroll, J.M., Kellogg, W.A., & Rosson, M.B. (1990). The Task-Artifact Cycle. In J. Carroll (ed.) *Designing Interaction: Psychology at the Human-Computer Interface*. New York: Cambridge University Press.

Conklin, J. & Begeman, M. (1987) gIBIS: A Hypertext Tool for Team Design Deliberation. In *Hypertext'87 Papers*. 247-251. Chapel Hill, NC.

Fischer, G., McCall, R., & Morch, A.I. (1989). Design Environments for Constructive and Argumentative Design. *CHI'89*. 269-275. Austin, TX.

Fischer, G., Lemke, A.C., Mastaglio, T., & Morch, A.I. (1989). Using Critics to Empower Users. *CHI'90*. 337-348. Seattle, WA.

Hill, W.C. (1989). The Mind at AI: Horseless Carriage to Clock. *AI Magazine*, 10(2): 28-41.

Lenat, D.B. & Guha, R.V. (1990). *Building Large Knowledge Based Systems*. Reading, MA: Addision-Wesley.

Terveen, L.G. (1991). Person-Computer Cooperation Through Collaborative Manipulation. Ph.D. Thesis. Dept. of Computer Sciences, The University of Texas.