Intelligent Assistance through Collaborative Manipulation[1]

Loren G. Terveen David A, Wroblewski Steven N. Tighe

AT&T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974
tervcen@research.att.com

US West
4001 Discovery Dr.
Boulder, CO 80303
davew@ us wcst.com

MCC
3500 W. Balcones Center Dr.
Austin, TX 78759
tighe@mcc.com

Abstract

This paper introduces, motivates, and illustrates an approach to the construction of intelligent assistance systems that we call *collaborative manipulation.* We show how a system can offer effective assistance through collaborative manipulation of objects in a shared workspace. We have developed this approach through experience with an intelligent knowledge editing tool, the HITS Knowledge Editor. We illustrate its effectiveness using scenarios taken from a user study.

1 Intelligent Assistance

Intelligent assistance is an active research field within AI [Chin, 1988; Lochbaum *et al.,* 1990; Lemke and Fischer, 1990; Miller *et al.,* 1990]. This research is motivated by several factors, including (1) the importance of collaboration in intelligent activity, (2) the scarcity of totally formalizable domains, and (3) people's need for help with increasingly complex computer applications.

Two key issues in the design of an intelligent assistance system are - *what is the role of the system in the inter action? and how is the system-user interaction managed!* Our work in building an intelligent assistant for the task of *knowledge editing* has led us to three design principles that address these issues. We call the approach characterized by these principles *collaborative manipulation.*

2 Collaborative Manipulation

We carry out our research within the paradigm of *cooperative problem solving systems* [Lemke and Fischer, 1990]. This approach begins from the premise that people and computers have vastly different strengths and weaknesses and that effective cooperation needs a division of responsibility based on the strengths of each party. Our contribution is to base system assistance on collaborative manipulation of objects in a shared workspace. The approach has three key aspects.

1. *Provide a* <u>*workspace*</u> *for joint user-system problem solving.* People at everyday tasks construct personalized work contexts that include task-relevant materials and *partial specifications of solutions* - think of a kitchen while someone is cooking or your workstation and desk while you debug a program or write a paper. In this paper, we focus on two properties of a workspace that are useful for an intelligent assistant: first, it provides access to users[1] *partial solutions,* enabling the assistant to compute advice in a timely manner, and second, the assistant can deliver significant aspects of its advice by manipulating objects in the shared workspace. We call the latter process *advertisement* [Wroblewski *et al.,* 1991].

2. *An effective role for an intelligent assistance system is that of a* <u>*design critic*</u> [Fischer *et* al., 1990]. In design, a person constructs an artifact meeting certain constraints - in knowledge editing, users construct knowledge structures that encode their understanding of a domain and fit in with the constraints of a knowledge representation system. A critic "looks over the shoulder" of users as they perform a task and offers advice occasionally. In our system, critics propose completions of unfinished objects, detect problems, and suggest additional issues.

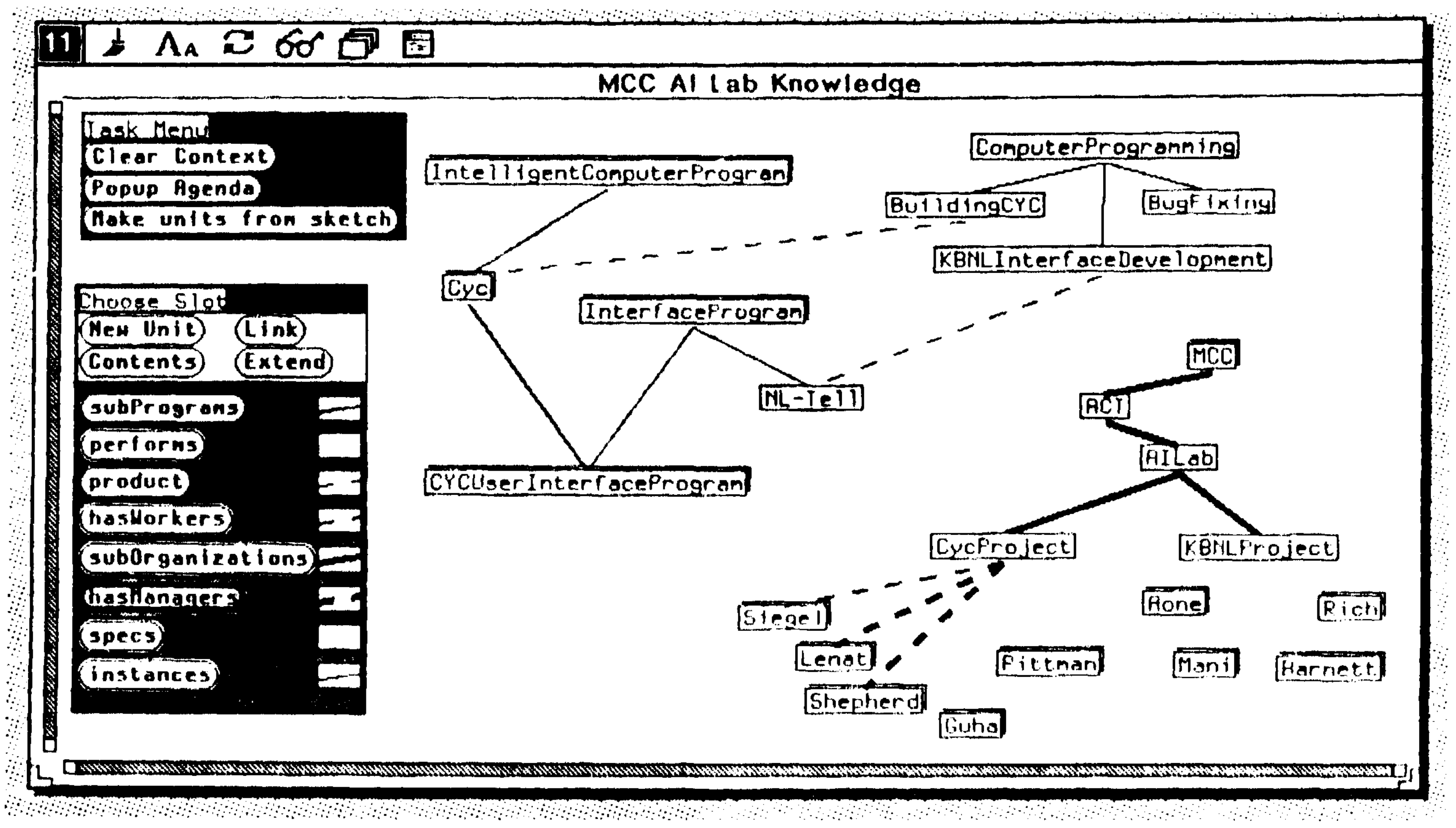
3. *The user-system interaction must be managed according to conventions that are appropriate to the abilities and roles of each party.* For example, a system should (1) avoid taking the initiative from users and forcing them to deal with advice at the convenience of the system, and (2) avoid imposing a fixed order of work on users. In our system, interaction is organized in terms of propose-critique-refine interchanges, in which the user always has the final say.

3 The HITS Knowledge Editor

Knowledge editing involves the entry, viewing, access, and maintenance of information in a knowledge base. Many systems make strong assumptions about the type and use of knowledge being entered [e.g. Kahn *et al.,* 1987; Musen *et al,* 1987] in order to guide users. Our approach, however, along with Murray and Porter [1990] is to assist users in the knowledge editing task without making such assumptions.

Representing knowledge in a knowledge base is a difficult task. People must articulate knowledge to a higher degree of precision than is required for everyday communication and must encode the new knowledge in harmony with existing knowledge and representational conventions. The HITS Knowledge Editor (HKE) assists users in this task. HKE is a browsing/entry interface to CYC [Lenat and Guha, 1990]. HKE embodies an analysis of knowledge editing into six sub-activities [Terveen, 1991].

[1]This work was done while all three authors were at the MCC Human Interface Lab.



Double lines around an icon indicate that an object by this name exists in the knowledge base, e.g., Mac exists but ACT does not. The same convention is used for slot buttons in the Choose Slot menu: e.g., performs exists in the KB but product does not. The Choose Slot menu also functions like the legend of a map. Each slot has an associated line pattern, e.g., subOrganizations is represented by a thick solid line. The icons for two objects related by one of these slots are linked by the appropriate line pattern, e.g., MCC and ACT are related by the subOrganizations slot, so their icons are linked by a thick solid line.

Figure 1 - A sketch

We focus here on three activities that comprise knowledge *entry,* since this where HKE offers the most assistance.

- During *specification,* users sketch out new knowledge via a direct manipulation interface.
- During *incorporation,* the system merges the specification into the knowledge base and detects problems and issues.
- During *repair,* the system presents the problems and issues it has detected and works with the users to resolve them.

We illustrate how HKE assists in these three activities with a scenario taken from a user study [Terveen, 1991]. Pairs of subjects were asked to represent knowledge about the structure of their organization (the Artificial Intelligence or Human Interface Laboratory at MCC), such as researchers and their areas of expertise, projects, and software systems.

Some CYC terminology is necessary to understand the illustration. Objects in CYC are called *units.* We use typewriter font to indicate units, e.g., MOC, Terveen, Worker. Slots are first class units. The domain of a slot is recorded on its makesSenseFor slot and the range of a slot is recorded on its entry IsA slot. By convention, slot names begin with lowercase letters, e.g., hasWorkers and instanceOf. We use predicate argument form to refer to assertions in the knowledge base, e.g., hasWorkers(MCC, Terveen) means that Terveen is a filler of the hasWorkers slot of MCC We use the notation unit.slot to refer to the value or values of a particular slot of a particular unit, e.g., MOC.hasWorkers represents the set of workers at MCC.

3.1 Specification

Our user studies have shown that knowledge representation typically begins with a small group of people sketching out key objects and relationships on a piece of paper or a whiteboard. In HKE, users specify new knowledge by sketching a graph of objects and their relationships using a direct manipulation interface. Users can sketch only those objects and relationships that are of most immediate interest to them - they do not have to satisfy CYC's requirements for well-formed units immediately.

While sketching out new information, users often need to explore the knowledge base for existing information that is relevant to their task. HKE provides browsing methods to do this. Relevant objects can be collected in the sketch; thus, users can create a context for solving their problem as part of the problem solving process [Suchman, 1983]. Figure 1 shows an intermediate point in the specification activity of one pair of subjects.

3.2 Incorporation

When users are satisfied with their specification, they request HKE to incorporate it into the knowledge base. While doing so, HKE applies rules to each assertion that (1) *infer* additional assertions, (2) discover *constraints* between objects and (3) detect *troubles* or *suggestions* that apply to an object.

3.2.1 Inferences and Constraints

HKE infers required information that users have not specified based on how objects are used in the sketch. For example, in figure 1 the users introduced a new slot, product, without specifying either its domain and range. However, they did state that Cyc was a product of BuildingCYC, and Cyc already is known to be an instance of IntelligentCcrrputerPrograra Therefore, the system inferred that the range of p roduct was IntelligentCcxrputerPrograra

Sometimes inferences can be made only on the basis of non-local information in the sketch, i.e., a value inferred on the basis of one assertion may affect an object in another assertion. HKE supports this by using *constraints.* For example, suppose HKE processes the assertions

• product(KBLInterfaceDevelcpnrent,NL-Tell) and
• product(BuildingCYC, Cyc),

in that order. When the system processes the first assertion, it does not know anything about the objects product, NL-Tell, and KBNLInterfaceDevelopment. However, based on this assertion it creates two constraints:

• productjnakesSenseFor 6 KBNLInterfaceDevelopment. instanceOf: the domain of product must a class which KBNLInterf aceDevelopment is a member of.
• product. entryIsA E NL-Tell. instanceOf: the range of product must be a class which NL-Tell is a member of.

When HKE processes product(buildingCY/C, Cyc) and infers

entryIsA (product, IntelligentComputerProgram),

maintaining the second constraint enables HKE to infer that NL-Tell is an instance of IntelligentComputerProgram When HKE determines that KBNLInterfaceDeveloprrent is an instance of CorrputerPrograirming, the first constraint will be maintained with similar results. Figure 2 summarizes part of the inference process just described.

HKE records the justification for each inference that it makes. Users can access the justification during the repair activity as a resource in deciding whether to accept or modify the system's inference (see figure 4). For example, they might decide that the range of product should be a more general class than IntelligentComputerPrograiTL

There are several reasons why the type of inferencing that HKE does is particularly useful. First, it reduces what users have to know and decide. For example, novice users may not know that they have to specify the domain and range of a slot, but HKE can make consistent guesses about this information based on how they have used the slot in their sketch. Second, no options are taken away from users: they are still free to modify the values that the system has inferred. In fact, arguably the most important feature of system inference is that it can draw users' attention to issues that they had not considered. Furthermore, the justification for an inference is available to users as they decide whether to accept it, and, if they decide to seek an alternative answer, the system provides follow-up options that guide users in exploring the space of alternatives. This also facilitates a kind of learning: users can become aware of both new issues and ways to resolve the issues.

3.2.2 Troubles

Every assertion stated by the users is examined to see if it is inconsistent with information already in the knowledge base - this is a *trouble.* Some troubles are relatively simple and localized. For example, the most common trouble encountered in the user studies was that users asserted a relationship between two objects, and the objects did not satisfy the domain or range constraints on the relationship. In figure 1 the users asserted hasManagers(CycProject, Lenat). However, the range of hasManagers is Manager and Lenat is not an instance of Manager, so HKE detects a trouble with this assertion.

Other troubles result from inconsistencies with inherited or inferred information. For example, in figure 1 the users asserted that CYCUserInterfaceProgram was an instance of InterfacePrograia Through inheritance, this would have the effect of making CYCUserInterfaceProgram an instance of the class I ndividua1Object. However, CYCUserInterfaceProgram already is known to be a derived instance of Collection, and Collection and individualObject are declared to be *mutually disjoint,* i.e., no object can be an instance of both classes. Therefore, HKE detects a trouble with the users' assertion (figure 5).

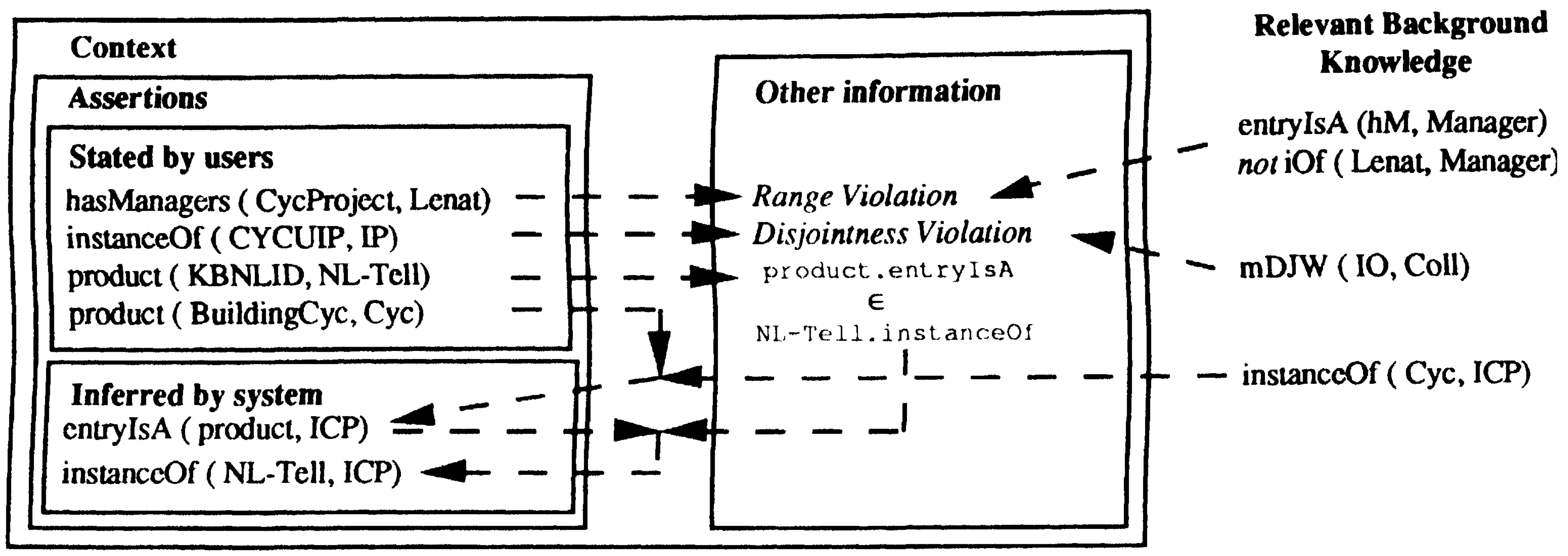
When the system detects a trouble with an assertion from the users' specification, it does not attempt to add that assertion to the knowledge base nor does it immediately engage users in a dialogue to repair the trouble. Instead, it creates a resource for repairing the trouble, associates it with the assertion, and advertises the trouble through objects in the work context (see next section for discussion).

There are a number of reasons why HKE docs not attempt to repair troubles automatically. First, HKE often knows alternative repair methods that it has no means of selecting among. Second, there may be repair methods that HKE does not know about - for example, radical changes to the class hierarchy can drastically change the set of legal assertions - and if users are skilled enough to think of such actions and perform them, they should be able to. Finally, sometimes the repairs that HKE offers are "dangerous" - they could have large ripple effects throughout the knowledge base - and should be done only after careful consideration. Therefore, all troubles are advertised to users, and the system assists in deciding how to repair troubles but docs not do so automatically.

3.2.3 Suggestions

Suggestions arc issues that deserve possible investigation, but do not prevent any part of the users' specification from being incorporated into the knowledge base. In figure 1, for example, the users defined a new class, CcrrputerProgramming, with three instances BugFixing, BuildingCYC, and KBNLInterfaceDevelopment. They later used the slot performs to relate different people to the three instances, e.g., performs (Lenat, BuildingCYC). The range constraint on performs let the system infer that the three objects were instances of the class PerformingAnActioa Rather than each of these objects being instances of both ComputerPrograrrning and PerformingAnActioa it might be preferable to make CcrrputerProgramming a subclass of PerformingAnAction. HKE therefore creates a suggestion that users consider this issue (see figure 4).

During incorporation, the system constructs a context (figure 2 shows selected parts) that includes the assertions from the users' specification and those inferred by the system. All the assertions concerning an individual object arc organized into a checklist [Terveen and Wroblewski, 1990]. The context is annotated with other information including troubles, suggestions, and constraints. Since the context includes many items that must be acted on, (e.g., troubles must be resolved, suggestions should be deliberated, and inferences can be verified or modified), it is essential that the system's representation of the context is shared with the users. This is the topic of the next section.



Abbreviations: CYCUIP – CYCUserInterfaceProgram, KBNLID – KBNLInterfaceDevelopment, hM – hasManagers
ICP – IntelligentComputerProgram, mDJW – mutuallyDisjointWith

Figure 2: Part of the context built by the system during incorporation

3.3 Repair

During repair, user and system jointly explore the consequences of the issues raised by the system during incorporation. In responding to system recommendations, users refine their initial conceptions of their domain based on the interaction between new and existing information.

The sketch and checklists serve as media for the system's recommendations. After incorporating a sketch, if the system detects troubles with an object, it displays that object in reverse video, and, if the system has suggestions about an object, it grays that object (see figure 3). Thus, the system uses the materials of the work context to advertise those objects that require further user attention.

Users repair an object by interacting with its checklist. The checklist advertises aspects of the object that require more attention. Figures 4 and 5 show the checklists for ComputerProgramming and CYCUserInterfaceProgram and an assistance resource accessible from each checklist.

Conventions used in checklists include the following. Reverse video indicates a trouble - e.g., the object interfaceProgram on the instanceOf slot of CYCUserInterfaceProgram (figure 5). Italics indicate an inferred value - e.g., the object Collection on the instanceOf slot of ComputerProgrartming (figure 4). A box around an object indicates that the object is incompletely specified - e.g., the object product on the canHaveSlots slot of CorrputerProgramning (figure 4). A balloon icon with text indicates a suggestion (figure 4).

An assistance resource is associated with each object that requires user attention. The suggestion associated with CorrputerProgramming and the trouble associated with CYCUserInterfaceProgram discussed in the previous section arc shown in figure 4 and 5, respectively. Since the resources that explain troubles or suggestions or inferences are made persistent through association with objects in the workspace, users can interact with them, turn their attention elsewhere, then revisit them later.

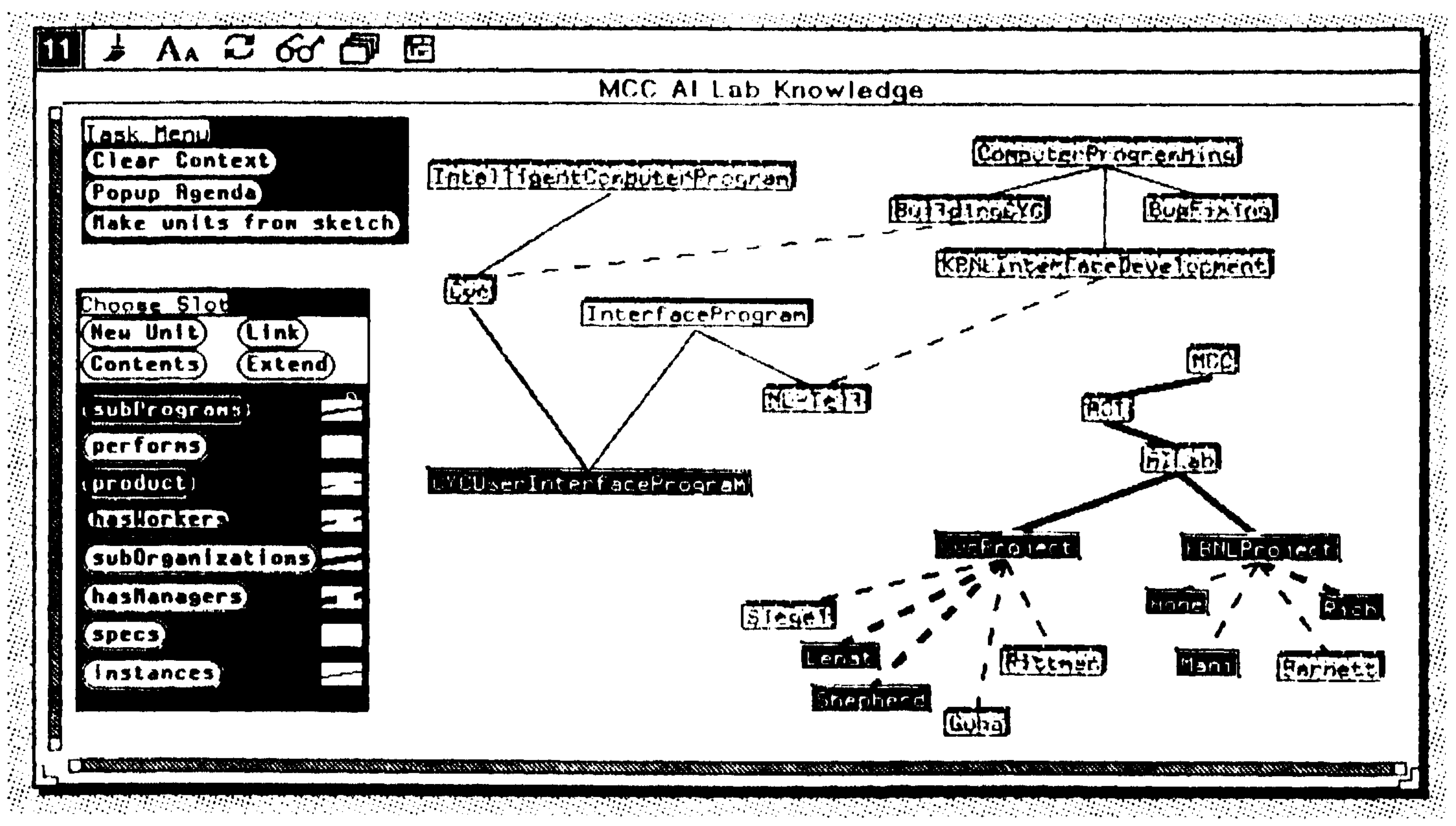
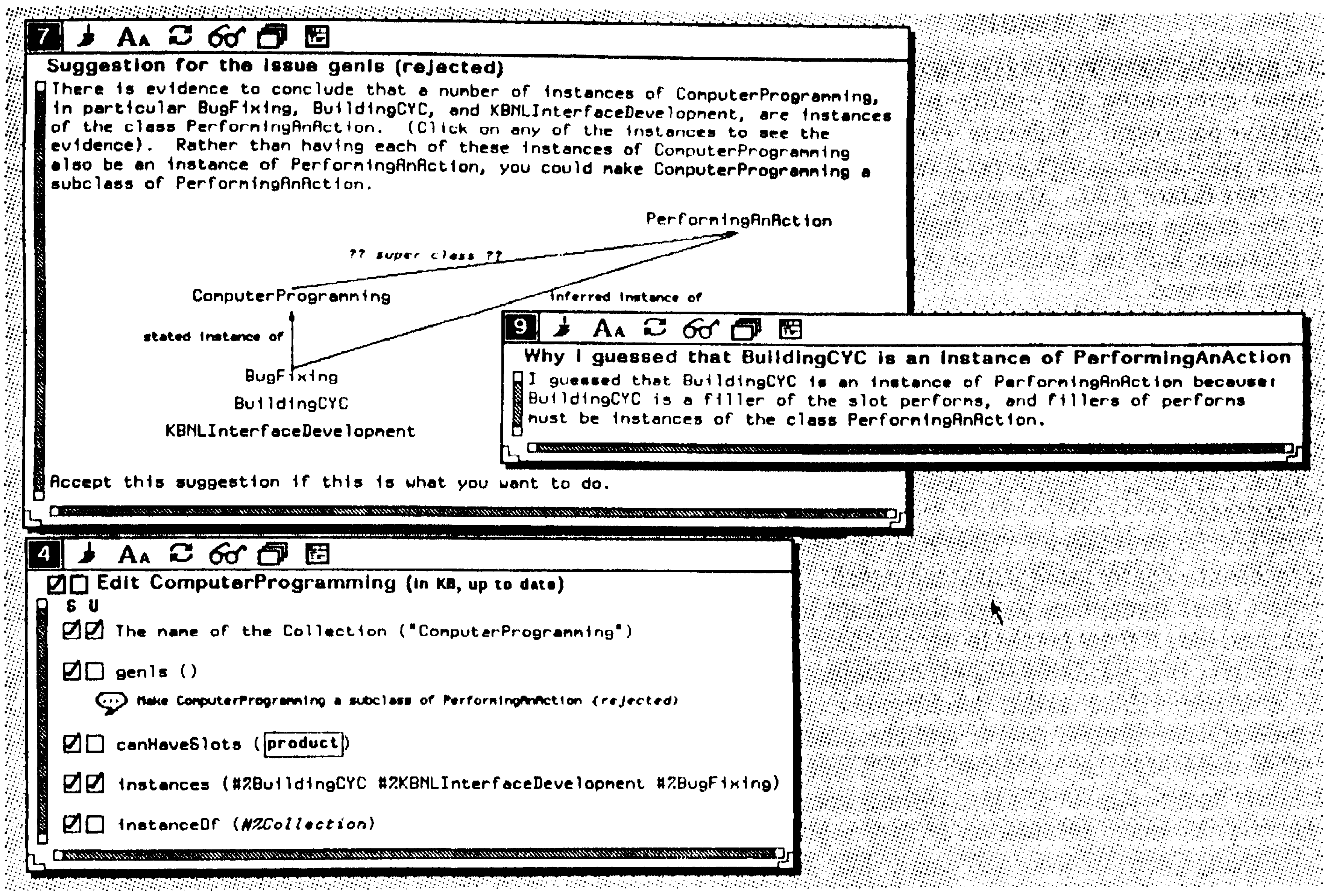


Figure 3: The sketch after incorporation



Suggestion for the issue genls (rejected)

There is evidence to conclude that a number of instances of ComputerProgramming, in particular BugFixing, BuildingCYC, and KBNLInterfaceDevelopment, are instances of the class PerformingAnAction. (Click on any of the instances to see the evidence). Rather than having each of these instances of ComputerProgramming also be an instance of PerformingAnAction, you could make ComputerProgramming a subclass of PerformingAnAction.

Accept this suggestion if this is what you want to do.

Why I guessed that BuildingCYC is an instance of PerformingAnAction

I guessed that BuildingCYC is an instance of PerformingAnAction because: BuildingCYC is a filler of the slot performs, and fillers of performs must be instances of the class PerformingAnAction.

☑☐ **Edit ComputerProgramming (in KB, up to date)**

☑☑ The name of the Collection ("ComputerProgramming")

☑☐ genls ()

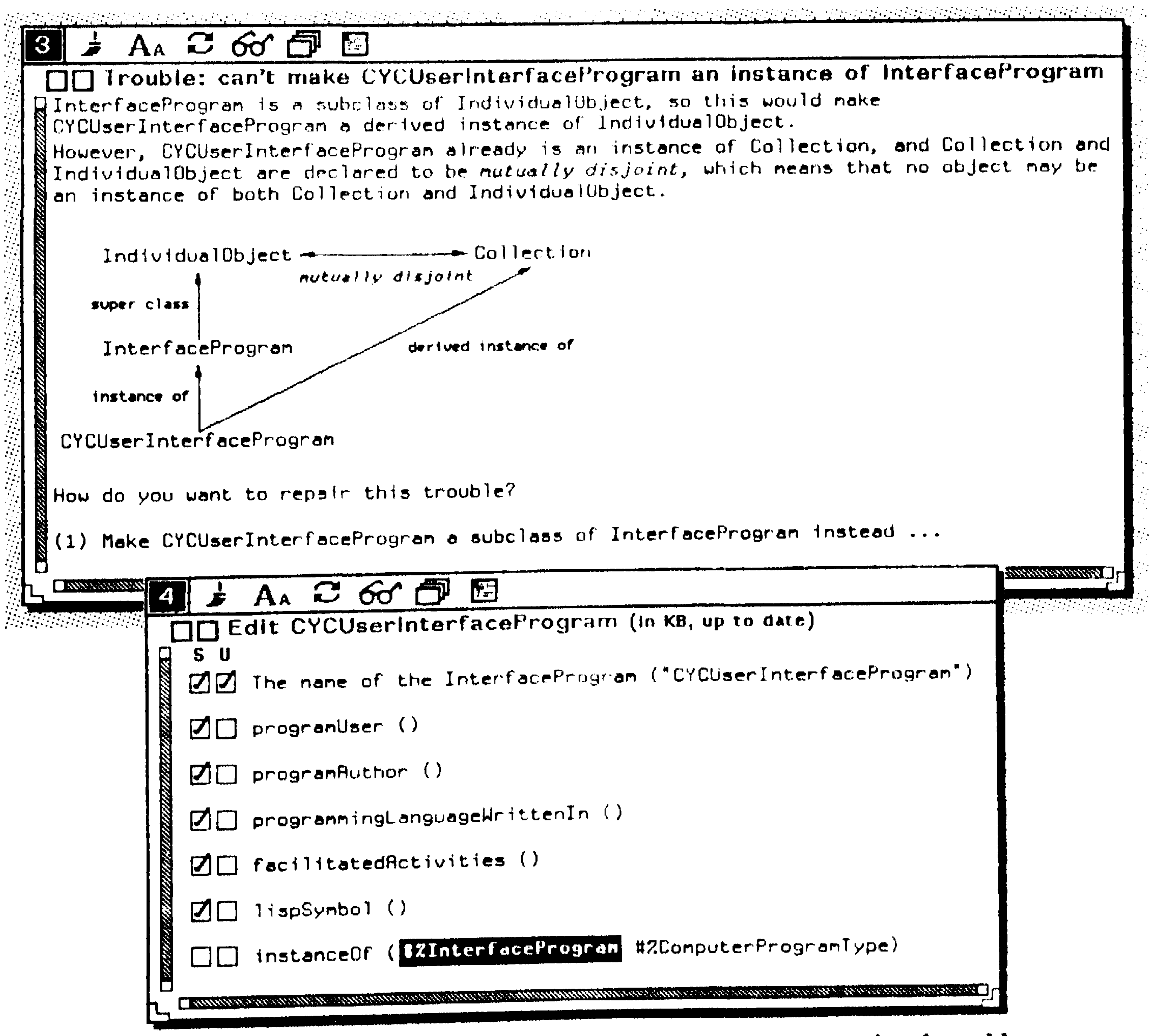
Make ComputerProgramming a subclass of PerformingAnAction *(rejected)*

☑☐ canHaveSlots (product)

☑☑ instances (#%BuildingCYC #%KBNLInterfaceDevelopment #%BugFixing)

☑☐ instanceOf (*#%Collection*)

Figure 4: Checklist for ComputerProgramming, with an associated suggestion



☐☐ **Trouble: can't make CYCUserInterfaceProgram an instance of InterfaceProgram**

InterfaceProgram is a subclass of IndividualObject, so this would make CYCUserInterfaceProgram a derived instance of IndividualObject.
However, CYCUserInterfaceProgram already is an instance of Collection, and Collection and IndividualObject are declared to be *mutually disjoint*, which means that no object may be an instance of both Collection and IndividualObject.

How do you want to repair this trouble?

(1) Make CYCUserInterfaceProgram a subclass of InterfaceProgram instead ...

☐☐ **Edit CYCUserInterfaceProgram (in KB, up to date)**

☑☑ The name of the InterfaceProgram ("CYCUserInterfaceProgram")

☑☐ programUser ()

☑☐ programAuthor ()

☑☐ programmingLanguageWrittenIn ()

☑☐ facilitatedActivities ()

☑☐ lispSymbol ()

☐☐ instanceOf (#%InterfaceProgram #%ComputerProgramType)

Figure 5: Checklist for CYCUserInterfaceProgram, with an associated trouble

4 Results

We have performed user studies [Terveen, 1991] that support our claims concerning the utility of the collaborative manipulation paradigm. Subjects were given the task of using either HKE or an earlier generation knowledge editing tool, the Unit Editor (UE) [Shepherd, 1988] that does not embody the design principles described in this paper. The studies illustrate both the benefits of the principles and the cost of their absence.

1. *A workspace for joint user-system problem solving is essential.* HKE's sketches allow users to collect relevant objects, thus creating *personal organizations* of knowledge relevant to the task at hand, rather than adhering to the logical organization of the knowledge base. Sketches give critics access to partial solutions, enabling the delivery of timely assistance.

The UE has no workspace. Users had to track relevant objects by memory or by using paper and pencil. Therefore, even expert users sometimes forgot significant unresolved issues because they did not persist in the interface.

2. *An effective role for an intelligent assistance system is that of a critic.* The critic paradigm exploits the complementary strengths of people and computers. Users know what they want to represent. HKE knows about representing knowledge in CYC. During specification, users can state as much information as they want to or arc able to, ignoring (what to them are) details like the domain and range of a slot. During incorporation, HKE draws on its expertise about knowledge editing to detect issues that are raised by merging the specification into the knowledge base.

HKE embodies much expertise about knowledge editing; the UE is an entry tool only, with no assistance component. Experts were able to perform equally well with either tool, since they had mastered knowledge of what issues to consider, how and when to resolve them, and the form in which CYC requires information to be stated. Novices did not possess such expertise and ran into significant problems using the UE. For example, sometimes they could not repair problems, they used a limited set of repair methods, and they never considered issues that experts did (but that HKE would raise).

3. *Use appropriate conventions for the user-system interaction.* Because assistance in HKE is *advertised* through objects in the workspace, users always retain the initiative. Issues for consideration always are presented in parallel. Users choose which issues to consider and the order in which to consider them.

In comparison, the UE utilizes sequential menu or query-based dialogues; thus, users sometimes had to consider issues of secondary importance or risk losing track of the issues completely. For instance, HKE allows users to introduce new objects simply by adding them to the sketch. Later, during incorporation and repair, required information not supplied or inferred is advertised as issues to be resolved. In the UE, users are forced to define each object before it is used. For example, users may have to suspend work on stating the assertion hasWorkers (MX-HI-Lab, Terveen) to ensure that hasWorkers, MCX:-HI-Lab, and Terveen are well-formed objects. Thus, the UE increased rather than decreased the cognitive load on the users.

In summary, the significant contribution of our research is to illustrate a method for delivering assistance that exploits the interactive potential of direct manipulation technology. In our view, delivery of intelligence in the interface is of primary importance, and the method of computing advice is secondary. Although for the purposes of exposition we have characterized our work in terms of three distinct design principles, in practice, the principles interact, and the power of our approach derives from this interaction. It is a workspace *combined with* the intelligence of critics *combined with* the delivery of assistance by advertising issues that make HKE an effective, coherent system.

References

Chin, D.N. 1988. Intelligent Agents as a Basis for Natural Language Interfaces. Ph.D. Thesis. Computer Science Division, The University of California at Berkeley.

Fischer, G., Lemke, A.C., Mastaglio, T., & Morch, A.I. 1990. Using Critics to Empower Users. In *Proceedings of CHI'90.* Seattle, WA.

Kahn, G.S., Breaux, E.H., DeKlerk, P., & Joseph, R.L. 1987. A Mixed-Initiative Workbench for Knowledge Acquisition. *International Journal of Man-Machine Studies,* 27:167-179.

Lemke, A.C., & Fischer, G. 1990. A Cooperative Problem Solving System for User Interface Design. In *Proceedings of AAAI'90.* Boston, MA.

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

Lochbaum, K.E., Grosz, B.J., & Sidner, C.L. 1990. Models of Plans to Support Communication: An Initial Report. In *Proceedings of AAAI'90.* Boston, MA.

Miller, J.R., Hill, W.C., McKendree, J., McCandless, T., & Terveen, L.G. 1990. IDEA: from Advising to Collaboration. SIGCHI Bulletin. 21(3): 53-58.

Murray, K.S. & Porter, B.W. 1990. Developing a Tool for Knowledge Integration: Initial Results. *International Journal of Man-Machine Studies,* 33:373-383.

Musen, M.A., Fagan, M.L., Combs, D.M., & Shortliffe, E.H. 1987. Use of a Domain Model to Drive an Interactive Knowledge-Editing Tool. *International Journal of Man-Machine Studies,* 26: 105-121.

Shepherd, M. 1988. Tools for Adding Knowledge to the CYCLSKB. Technical Report ACA-AI-068-88. MCC. Austin, TX.

Suchman, L. 1983. Office Procedures as Practical Action: Models of Work and System Design. *ACM Transactions on Office Information Systems.* l(4):320-328.

Terveen, L.G., & Wroblewski, D.A. 1990. A Collaborative Interface for Browsing and Editing Large Knowledge Bases. In *Proceedings of AAAI'90.* Boston, MA.

Terveen, L.G. 1991. Person-Computer Cooperation through Collaborative Manipulation. Ph.D. Thesis. Department of Computer Sciences. The University of Texas at Austin.

Wroblewski, D.A., McCandless, T.P., & Hill, W.C. 1991. DETENTE: Practical Support for Practical Action. *Proceedings of CHI'91* New Orleans, LA.