International Workshop on Artificial Intelligence for Industrial Applications 1988

TOOLS FOR ADDING KNOWLEDGE TO THE CYC LSKB

Mary Shepherd
Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, Texas 78759
USA

Abstract

The CYC Project is a ten-year research project now in its fourth year at MCC in Austin, Texas. Its goal is to build a large, real-world knowledge base. This type of knowledge base is necessary if AI is going to overcome the problems in knowledge acquisition and of brittleness that currently limit the strength and the breadth of AI programs. Knowledge is entered into the system by knowledge enterers whom we call "Cyclists," using the CYC user interface. The CYC interface is not designed for the naive or casual user, rather it is designed to enhance the productivity of the expert.

The paper traces the history of the interface, and explains how the current tools allow the Cyclist to add knowledge effectively. It also discusses the knowledge server architecture which allows several Cyclists to work on the knowledge base simultaneously.

Knowledge Representation

Information in CYC is stored as "frames." Each frame, or "unit," is comprised of slots and the values for those slots. There are separate units that represent each kind of slot, function, and problem solving method, as well as each domain-level or object-level concept in the system.

This means that CYC is self-referential in the same way that a dictionary is self-referential. However, unlike a dictionary, the CYC system will have a unit for each separate meaning of each word. For instance, consider the English word "bank." There will be a unit for each of the concepts: monetary bank, river bank, cloud bank, the bank of a curve, a bank in billiards, etc. This type of disambiguation enables us to cleanly represent knowledge about each distinct concept.

Also, unlike a dictionary, there will be a CYC frame for each important concept even if no single word names it. Already, many CYC concepts can not be described by a single word in English—or any other natural language (e.g., CompositeTangible&IntangibleObject, TheUnitedStatesDuringAnElectionYear).

We envision CYC circa 1994 as containing all the knowledge that is implicitly and explicity found in a one volume desk encyclopedia. This includes not only the information explicitly stated in each article, but, more importantly the information that the person who wrote that article expected that the reader would already know.

If one saw the statement: "Hirohito is the emperor of Japan," then one would consider the following questions: Has Hirohito always been Emperor of Japan? Has the Emperor of Japan always been Hirohito? Will Hirohito always be Emperor of Japan? Has Japan always had an Emperor? Is Hirohito a citizen of the land that comprises Japan, or the Japanese Government? As a person with knowledge of the world, you can answer these questions. In order for CYC to answer them the system must be provided with information about Japan, Hirohito, emperors, countries, the average life span of a human being and a country and a form of government, and other such obvious facts not specifically stated in an encyclopedia.

Some of this information is simple common sense, or consensus reality, which the writer of the article felt safe in assuming the reader would know. For instance, Japan is comprised of the people who are citizens of Japan, the land that comprises Japan, and the Government of Japan. Each of these facets change over time. The number of Japanese citizens changes daily. There have been times when Japan has controlled more land and times when it has controlled less. The Government of Japan is not the same now as it was 100 years ago. To represent all of this we began by creating a unit named Japan. That unit has references to every other unit that has a relationship to Japan. For instance, two of the entries in the value of the capitalCity slot on the unit Japan are CityOfEdo and CityOfTokyo. One of the historicIncarnations of the CityOfEdo is CityOfEdo1603 and one of the historicIncarnations of the CityOfTokyo is CityOfTokyo1988. Therefore, if asked a question about the capital of Japan in the year 1603, CYC would know the city being referred to was Edo. If asked about the Emperor of Japan, CYC would need to know—or guess—which year the question refers to.

In short, CYC is based on concepts, not on words. Most of the units in the system will deal with complex multi-word concepts. We have found this to be a difficult or surprising notion for many people to grasp. Therefore, it has been much more difficult than we had assumed it would be to find knowledge enterers: people who are able to look at the world and translate it into CYC concepts. It has been especially difficult to identify people who are able to recognize the consensus reality (implicit facts) assumed by the author of an article so that this essential knowledge could be included during knowledge entry.

Knowledge Entry

Cyclists interact with the system using the Unit Editor. The Unit Editor (UE) is a context-driven browser/editor for the CYC Knowledge Base (KB). There are more than 50 possible editing operations in the system at present which are invoked in several different ways: menu choices, keyboard-accelerators, control keys or mouse clicks, and combinations of keyboard/control keys and mouse clicks. For the convenience of the Cyclists, there is more than one command for almost every operation in the system; for example, typing `:rename`, hitting CONTROL-r, and clicking the mouse on the **RENAME UNIT** option in the command menu all do the same thing. UE allows the Cyclist to view or edit the existing set of units and slots, with the capability of saving locally (on a satellite machine) or globally (on the Knowledge Server) any changes made to the KB.

Each Cyclist has a Symbolics lisp machine consisting of a 1024 X 1024 bit mapped Black & White monitor, a color monitor, 30 megabytes (6 megawords) of RAM memory, and 360 megabytes of fast disk, plus access to gigabytes on a central Symbolics machine used as a file server. While this configuration is sufficient at present, we anticipate that we will saturate our machines in the near future. Fortunately, by increasing the amount of memory and disk in each machine, upgrading to the forthcoming Symbolics Ivory machine, we hope to keep ahead of the hardware limitations for as long as possible.

The Unit Editor can be thought of as comprising three separate, asynchronous processes: UE (B&W unit editor), MUE (a "museum room" editor on the color or B&W monitor), and the Background processes.

UE, the basic B&W frame editor, is where Cyclists spend most of their editing time (see figure 1). Cyclists may choose between preset display configurations to accommodate the dis-

88CH2529-6/88/0000-0365 $1.00

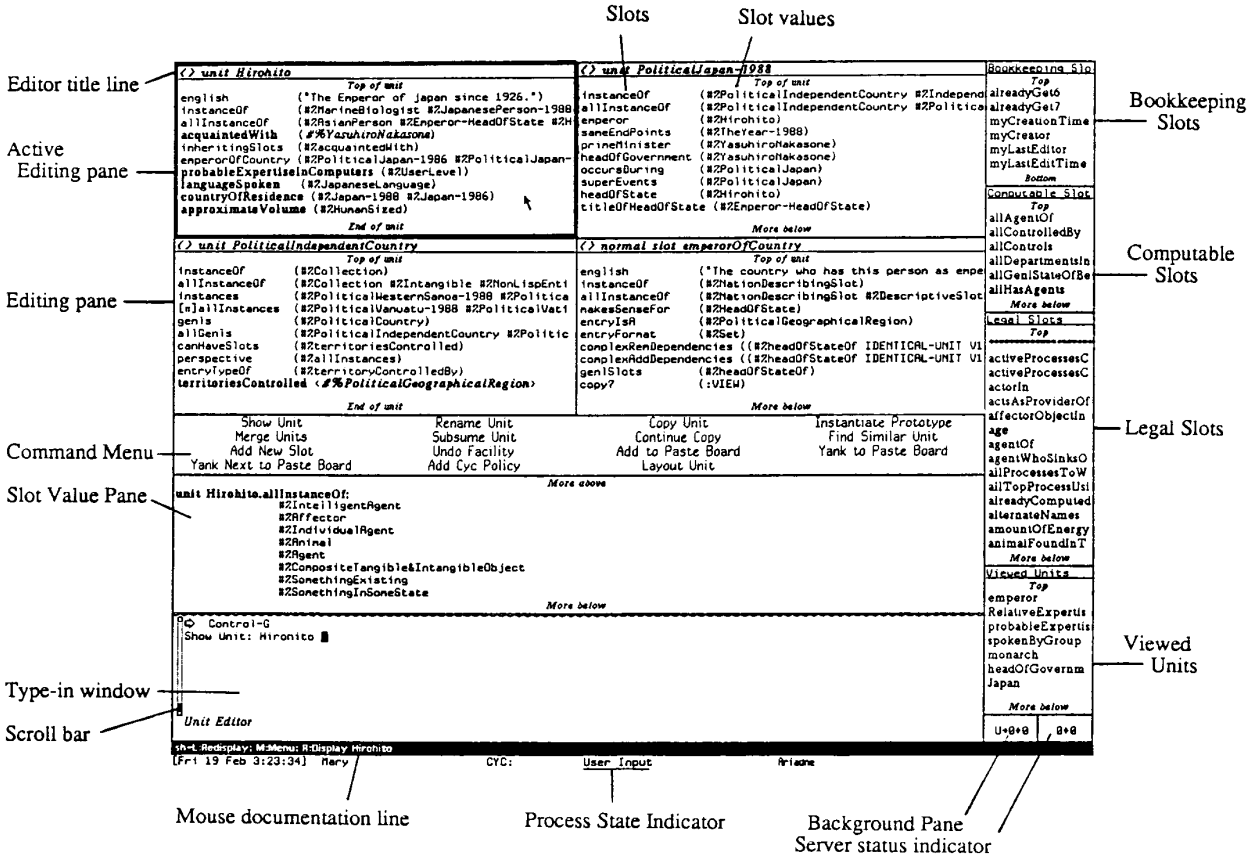

FIGURE 1. THE DEFAULT UE CONFIGURATION

play to their needs and the task at hand.

The graphical unit editor is called MUE, which stands for Museum Editor. While UE is text-oriented, MUE is spatially oriented. This editor gives Cyclists a graphical representation of how a chosen unit relates to other units. Each unit displayed is shown as a labelled rectangle, and related units—each represented recursively in the same way—are located at a particular position within it. For example, all specializations of the chosen unit are located near the bottom of the labelled rectangle (see figure 2). Color is used to reinforce the type of relationship being represented (e.g., all the specializations will not only be located near the bottom of their enclosing rectangle, but on a color monitor each will also have a blue border). One can run MUE on the B&W screen, but one then misses the "feel" of running it in color and (even more importantly) one misses having the extra screenful of "real estate."

The Knowledge Server (KS) is the subsystem which coordinates among all the n Cyclists using the system at the same time. Each user machine or "satellite" has a full version of CYC, and of the CYC KB. Each editing operation performed at a satellite is sent to the server machine, which then broadcasts it to all the other satellites; the server machine also adjudicates among the satellites in case of collisions.

The Background is a process that propagates the side effects of each top-level editing operation. If one adds Hirohito as a new instance of JapanesePerson, UE and MUE will instantly shift to reflect that. Then the background will start adding inverse links (e.g., from Hirohito to JapanesePerson), running forward inheritances (e.g., assigning default values for various slots of Hirohito, such as languageSpoken: JapaneseLanguage), and forward rules and demons. Only after the background is finished with all side effects, and if no errors have occurred on the satellite, will the operation actually be sent to the KS. If one edits rapidly, and there are many ramifications to one's actions, the queue of background operations can grow, and the system can lag noticably in propagating all the ramifications and ripples of those operations. The main reason for the existence of the background process is to allow the Cyclist to continue entering information without feeling the delay caused by the propagations.

When other Cyclists' operations are received from the server at one's satellite, they are placed on the background queue, just as though they were operations that were actually performed by the local UE.

There is a separate Background Window, which is not normally displayed. It signals strongly if it needs to report an actual error or ask the Cyclist questions which require immediate answers, but it will signal gently if it only needs to print a warning message. If it's more than a gentle signal, it means the Background process has halted. When this occurs, editing in the foreground UE can continue but operations will not be "assimilated" by the KB. They will simply be added to the queue until the background has been attended to. This allows the Cyclist to continue with the task at hand without the interruption that would necessarily occur if there were no background process. Another important reason the Background has its own window is to provide extra screen "real estate."

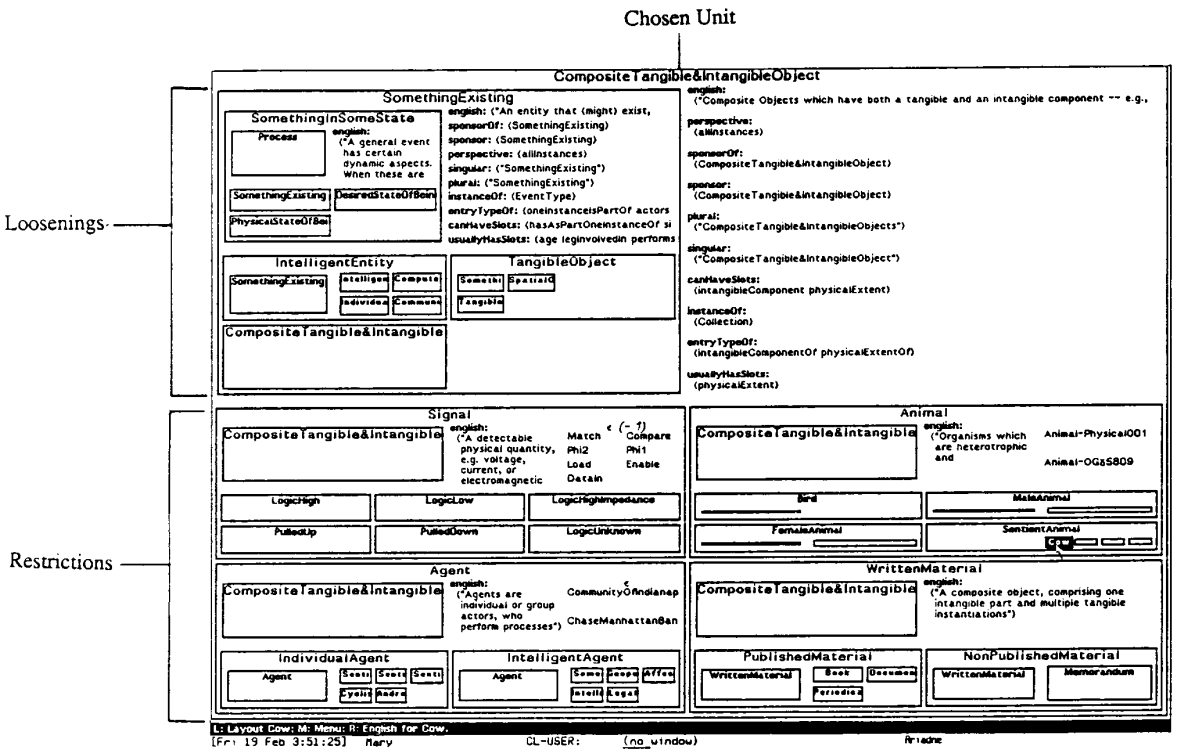

FIGURE 2. DEFAULT MUSEUM EDITOR CONFIGURATION

Unit Editor

Each of the four editing panes displays a separate CYC unit as rows of slot/value pairs. Most of the editing is done in the editing panes: viewing the existing state of a unit in the KB, clicking to make changes, and seeing the results of those changes appear. One of the editing panes is always the "Active Pane," and is so indicated either by being displayed with a thicker border, or in reverse-video (display choice is based on the value stored on the uePaneSelect? slot of the unit that represents the Cyclist who is logged into the machine).

The use of different type fonts increases the amount of information that can be conveyed to the Cyclist. For instance, if a slot name appears on a unit in bold font, that slot is inherited. If an entry on the value of an inherited slot appears in italics, the entry is local (it was entered by hand and not inherited); therefore, it will take precedence over a conflicting inherited value.

The *Server Attention Pane* shows two numbers separated by a diamond. The number on the right is the number of local operations queued and waiting for the Background to attend to them. The number on the left is the number of operations received from the server, if there are any, waiting for the local queue to empty so the Background can execute them and propagate their side effects.

The *Background Indicator Pane* has a dual role. The pane flips to reverse-video when the Background process has halted: it either encountered an error, or has a serious message to communicate to the Cyclist. The pane *grays-out* when the Background has some minor warning to communicate. The other role of the *Background Indicator Pane* is to provide information about the status of a satellite machine relative to the knowledge Server. The symbols in that pane tell the Cyclist if the KS is active, if the satellite machine is transmitting and/or receiving information, the number of minutes since the last communitation between the satellite machine and the KS, and the length of the KS operation queue.

The *Server Attention Pane* and the *Background Indicator Pane* provide much information to the Cyclist in a minimum of screen space. At a glance, the Cyclist knows the state of the satellite and server machines. Before this information was available in UE, Cyclists were always running back and forth between their satellite and the server to make sure the connection was still "up" and the machines were "talking" to one another.

Bookkeeping Slots are slots like myCreator, myLastEditor, myCreationTime, myLastEditTime, which are typically generated by the system, and which generally do not interest the Cyclist. While writing this paper I realized what a waste of screen space it is to have the *Bookkeeping Slots* pane: one usually wants to see *none* of them, or, more rarely, all of them displayed on a unit. The system was changed so that the *Bookkeeping Slots* will appear on a unit displayed in one of the editing panes on demand, and the space dedicated to the *Bookkeeping Slots* pane will be devoted to something more useful.

The *Computable Slots* for the selected unit are listed in that pane. These are slots whose values may be generated by the system on demand, but are not cached. They are typically computed from other, more primitive slots. Some examples are: allGeneralizations, allPartTypes, allSubProcesses.

The *Legal Slots* pane displays additional slots which the active unit *could* legally have but which it does *not* currently have. Since it takes a nontrivial amount of time (up to 10 seconds) to update the *legal slots* each time a new unit is selected, the default is for this pane to be blank. The Cyclist clicks on this pane to

have its contents computed and displayed. This makes editing much faster since many of the editing and browsing tasks do not involve seletcting or even seeing the list of legal slots for the current unit. There is a row of asterisks separating two alphabetical lists of slots in the *Legal Slots* pane. The slots listed above the asterisks in this pane are those slots that *usuallyMakeSenseFor* the active unit, and therefore are likely to be needed for that unit. The slots below the asterisks are those that *makeSenseFor* a unit, i.e., they are legal for the selected unit, but will not be used as frequently as those that appear above the asterisks. When creating a slot, the Cyclist tells the system the types of things that slot will *makeSenseFor* and *usuallyMakeSenseFor*. This pane will probably be moved to a new position soon. One thought is to place it vertically between the four editing panes to shorten the distance the mouse must travel to reach the desired slots.

The *Viewed Units* pane lists a stack of units that have been viewed during the current editing session but are not displayed in any of the editing panes at the present time. A simple mouse click will redisplay any of these units. The maximum size of the *Viewed Units* list is currently set at 25 unit names, but that may change. When UE was first written, we thought the Viewed Units would be very useful. However, over time we have come to realize that since a Cyclist generally edits one group of closely related units at a time, the name of the unit most likely to be needed is almost always on the screen somewhere (e.g., as an entry in the value of a slot on one of the displayed units). A command key/mouse click combination will display a unit whose name appears anywhere on the screen. Also, clicking on the <> symbol in the editor title line of a displayed unit flips to the unit previously displayed there. For these reasons then, the *Viewed Units* pane is not heavily used.

The *Command Menu*, is used almost exclusively by new Cyclists who have not yet learned the commands. Later, it can be eliminated from the display, to allow more space to the four editing panes. Since there is more than one command for almost every operation, Cyclists tend to develop "pet commands." For instance, when I want to create a new unit by copying another, I hit CONTROL-c. There are at least two other commands that will do exactly the same thing, but I never use them.

The *Slot Value* pane is where the value of a slot is displayed in its entirety at the Cyclist's request. This is useful since each slot is allotted only one line in an editing pane, and there are some slots whose entire value is not visible. A mouse click on a certain position in a value causes all the entries in that value to be printed in the *Slot Value* pane (the pane is scrollable if the value is very large). This pane also functions as the *Paste Board* display area. A paste board value can be created from any on-screen units, slots, whole values, single entries, etc. There are commands for modifying and editing values in the *Paste Board*, and other commands for adding the displayed value to a slot on a unit displayed in UE.

The *type-in* window is a Lisp Listener. This is the location on the UE screen where the Cyclist is able to type new entries, answer questions, etc. The UE default configuration provides a small *type-in* window in order to allow more space for the editing panes. However, the Cyclist can choose to have a very large *type-in* window and only two editing panes; this is useful when debugging or running a simulation which will cause a lot of type-out.

Museum Editor

The precursor of MUE was a directed graph (semantic net), but it was abandoned after about a year. The space became too crowded and updating and navigating the knowledge space became too slow and too difficult. We are not locked into MUE as the only graphical representation for the system: other configurations are in the planning stage.

MUE displays a chosen unit and its "neighborhood," as a set of labelled, nested boxes, each box representing a unit. On a system with a color monitor, the outline color for each box indicates the relationship of the unit represented by that box to the unit represented by the box outside of it. For instance, a cyan (light blue) box surrounds *specializations* of its surrounding unit, whereas a red box surrounds *generalizations* of its surrounding unit.

Figure 2 illustrates the MUE configuration used most frequently. It is organized so the units that are restrictions (units that are more specific in some way than the enclosing unit, e.g., specializations) of the unit are displayed in the lower half of the screen. The loosenings (units that are more general than the enclosing unit, e.g., generalizations) of the unit are displayed in the top left quarter of the screen. The top right quarter is divided in half vertically. The right half of this quarter-screen area displays a list of the entries in the value of the instances slot on the unit. The left half of this area displays a row of other miscellaneous slots that appear on the unit along with the values of those slots. This is useful when the Cyclist wants to get a global overview of the neighborhood of a specific unit.

Every unit name on the MUE screen is mouse sensitive; numerals and strings are not mouse sensitive. Many of the unit names displayed are truncated so they do not over-write other unit names, but when the mouse cursor is positioned over a truncated unit name, the name expands.

Each time a new unit is displayed as the chosen unit in MUE an entry is made on the *history list* which serves the same purpose as the *viewed units* list on UE. The *history list* becomes an array of memorized positions which can be rapidly traversed. So, at any moment there is (a) an array of past positions, (b) an "index" to the currently displayed position (which need not be at the end of the array, e.g., if the Cyclist was browsing back in time), and (c) an "index" to the next vacant spot in the array. The *history list* is organized with the most recently viewed unit at the top and the least recently viewed unit at the bottom.

MUE has editing capabilities that can be used without having to move the unit to UE. However, the ability to shift any displayed unit to UE is also present. That ability is useful for more complicated editing, or for viewing slot values not entirely displayed in MUE due to lack of space.

Using UE

I can not overstress the importance of the copy-and-edit process as the most efficient method of entering an enormous amount of new information in a knowledge base. CYC takes good advantage of this method in UE. One simple command begins the copy process that allows the Cyclist to create a large group of heavily interrelated units. It begins by copying the given unit—say Mammal to Reptile. Some slots' values will remain the same; (e.g., genls: Animal). Some will result in new units being created; new units analogous to those that were entries in the values of Mammal. For instance, the value of the slot zooHousing for Mammal is (SmallMammalHouse. LionHouse. ElephantHouse). The value for the zooHousing slot on the unit Reptile will be (ReptileHouse), and that new unit will be copied recursively from SmallMammalHouse. When copying a large group of interrelated units (e.g., Mammal), the system keeps track of those that have not yet been copied. The copy feature includes automatic text substitution suggestions based upon the information provided by the Cyclist as the copy process progresses. It also makes guesses at entries for slot values based upon the entries of that slot on other units of the same class. The certainty of the guess is reflected in the default choice for that entry on the copy menu (e.g., the "first choice" box indicates strong certainty, the "second choice" box indicates moderate certainty, and the "edit" box indicates strong uncertainty). The copy process may be suspended and then restarted from the point at which it was interrupted without "loss of information." This allows the Cyclist to back up to create other necessary units, to

check information on a unit not on the screen, or switch to the background window or MUE, and then continue.

One of the most important considerations when creating a new unit, is to copy a unit that is as close as possible to the new unit being created. This saves time, effort, and later corrections. For instance, to create a unit that represents the fruit "lemons," it would be best to copy a unit that represents another citrus fruit. This is because the unit that represents another citrus will have many of the same slots that would have to appear on the unit for lemons, and in most cases even the values for those slots would be the same. If there were no other units in the system that represented a citrus fruit, then another type of fruit might be copied.

One way to find a unit to copy is to use the find-similar-unit feature. This feature is also helpful when the Cyclist wants to know if there are any units in the system that have a slot with a particular entry in its value. Or if there are any units that have a slot with precisely the same value as that of a slot on a given unit. Or if there are any units that have all the same slots and values (not including *Bookkeeping Slots*) as a given unit, and if so, what are the names of those similar units.

One of the Cyclist's tasks is to differentiate units as much as possible. For instance, if the units Emperor-HeadOfState and King-HeadOfState had all the same slots and values, the system would have no way of differentiating them. It is then up to the Cyclist to add more information to each of those units until there is some information not shared.

There are times when a Cyclist discovers that two units have been created that are similar enough that they represent the same concept, and there should really only be one of them (e.g., the slots peopleIHaveMet and acquaintedWithPerson). For this reason there is a feature in the system that allows two units to be merged. The Cyclist decides which unit will remain, and CYC guesses which slots and values will appear on the remaining unit.

There is also a Subsume feature which allows one unit to be given the slot values of another similar, but more complete unit. In other words, if there are two units that would normally have many of the same slots, and many of the values of those slots would also be the same or of the same type, but one of the units is very skimpy and the other is fairly well developed, the subsume feature is the fastest way to fill out the skimpy unit.

When one begins to enter units about a new knowledge area, the first units that should be created are the prototypical units for all of the "things" that will be represented in that knowledge area. A prototypical unit is one that describes and represents a class of things. For instance if a Cyclist were entering information about bodies of water, the unit River, which is the prototypical unit for all rivers, would be created before the unit to represent the MississippiRiver. Because of this, there will be times when the Cyclist will have to create a unit (e.g., MississippiRiver) and yet there is no unit other than River close enough to use as a template from which to copy. To assist the Cyclist in such a situation, there is a feature which will create a new instance of a prototype with all the proper slots.

Oops!

With few exceptions, anything done can be undone using the undo facility. The exceptions involve lengthy operations that should be performed only after deliberation (e.g., merging two units or killing a unit). However, since there are so many commands in the system, it is generally easier and faster to use an editing command to undo a simple mistake than to use the undo feature. The only time I really want to use the undo feature is when I want to undo a lengthy operation like merging two units or killing a unit. Sigh.

If, for some reason, a unit must be deleted or renamed, these features propagate all side effects throughout the system. If completing a deletion would create a problem somewhere in the KB, the Cyclist is notified and suggestions are provided to eliminate the possible hazzard.

Knowledge Server

There are already 11 full time Cyclists in the CYC project and each of them has a Symbolics machine, upon which they enter knowledge. There is a version of the KB resident on each satellite machine. Each of the satellite sachines may be in one of three modes with respect to the knowledge Server: transmitting and receiving information; not transmitting but still receiving information; or not transmitting and not receiving information. The choice of mode is made by the Cyclist.

When information is entered on one of the transmitting satellites, that information is sent to the Server, which checks the transmission for consistency and legality. If the transmission passes all checks, the Server then performs related operations locally, and then sends them to all the satellite machines that are in receiving mode. This keeps all versions of the KB consistent.

When a Cyclist on a nontransmitting satellite decides to change modes and begin transmitting, the knowledge Server asks if all changes to the KB on that satellite should become part of the global KB and, therefore, be transmitted to the other satellites. If the Cyclist's answer is affirmative, the Server runs through the operations in order, checks each for legality, propagates side effects if there are any, and then sends the operations to all other receiving satellites. If the Cyclist's answer is negative, that satellite is isolated until the Cyclist reloads the system. Isolated in this context means the satellite will be in the not transmitting and not receiving mode. This is because while a satellite is not communicating with the Server, other satellites are, and the KB on the nontransmitting machines is inconsistent with them.

Conclusion

While not an interface for the novice, once mastered, the CYC user interface is a very useful and effective tool with which to enter large amounts of knowledge quickly and easily. The interface has grown and changed over time and will continue to do so (this paper caused two changes to be implemented). There are enhancements (e.g., a different type of graphical editor) being worked on now that will be implemented within a very short time. Since the Cyclists who write and implement UE are also using it, they are responsive to constructive ideas about how to improve it.

Bibliography

[1] D. Lenat, A. Borning, D. McDonald, C. Taylor, and S. Weyer, "Knoesphere: Expert Systems with Encyclopedic Knowledge," Proceedings of the Conference IJCAI-8, pp.167-169, 1984.

[2] D. Lenat, E. A. Feigenbaum, "On the Thresholds of Knowledge," MCC Technical Report Number AI-126-87, May 1987.

[3] D. Lenat, M. Prakash, M. Shepherd, "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks," The AI Magazine , vol. 6, no. 4, pp.65-85, 1986.