August 1985

Report No. STAN-CS-85-1068
*Also numbered KSL-85-20*

# GUIDON-WATCH: A Graphic Interface for Viewing A Knowledge-Based System

by

Mark H. Richer
William J. Clancey

**Department of Computer Science**

Stanford University
Stanford, CA 94305

# GUIDON-WATCH:

# A GRAPHICS INTERFACE FOR VIEWING

# A KNOWLEDGE BASE

Mark H. Richer
William J. Clancey

Stanford Knowledge Systems Laboratory
Department of Computer Science
701 Welch Road, Building C
Palo Alto, CA 94304

The studies reported here were supported (in part) by:

The Office of Naval Research
Personnel and Training Research Programs,
Psychological Sciences Division.
Contract No. N00014-85K-0305

The Josiah Macy, Jr. Foundation
Grant No. B852005
New York City

## Abstract

This paper describes GUIDON-WATCH, a graphic interface that uses multiple windows and a mouse to allow a student to browse a knowledge base and view reasoning processes during diagnostic problem solving. Methods are presented for providing multiple views of hierarchical structures, overlaying results of a search process on top of static structures to make the strategy visible, and graphically expressing evidence relations between findings and hypotheses. This work demonstrates the advantages of stating a diagnostic search procedure in a well-structured, rule-based language, separate from domain knowledge. A number of issues in software design are also considered, including the automatic management of a multiple-window display.

## 1. Introduction

An increasing number of Artificial Intelligence (AI) programs are implemented on high performance workstations with a bitmap display, a mouse, and a keyboard. The programming environment (usually a dialect of LISP) generally provides support for displaying multiple windows and using menus that can be selected with a mouse. Importantly, a programmer can also specify arbitrary regions of a window (e.g., text items) to be selectable with the mouse. This means that a user can invoke an action by pressing or releasing a mouse button while the mouse cursor is in a selectable region. These features make it possible to create a user interface that is efficient and easy to use for viewing and browsing a complex system.

The GUIDON project at Stanford University is investigating how knowledge-based systems can provide the basis for teaching programs. NEOMYCIN (Clancey and Letsinger, 1984), a medical consultation system, has been developed for this purpose. This paper describes GUIDON-WATCH, a graphic interface to NEOMYCIN that uses multiple windows and the mouse to allow a user to browse the NEOMYCIN knowledge base and view reasoning processes during a consultation. The results reported include methods for providing multiple views of a data base, techniques for illustrating dynamic processes including a search strategy, and some conclusions regarding automatic management of a multiple window display.

The paper is organized into the following sections:

- **Project Goals and Results to Date**
- **The Development of NEOMCYIN**
- **Description of GUIDON-WATCH**
- **Prior and Related Work in Graphic Interfaces**
- **Future Work**
- **Conclusions**

## 2. Project Goals

The ability to display and select information in several windows allows people to control and observe the behavior of an application program more easily. A graphic interface to a knowledge-based system can serve different kinds of users, including system designers, implementers, domain experts, students, and other end users.

In the GUIDON project, the end users will be medical students. We are currently collaborating with physicians and medical students to adapt NEOMYCIN, GUIDON-WATCH, and other programs for medical instruction. However, when this work began better tools were also needed to maintain the NEOMYCIN knowledge base and to debug program behavior. As a result, GUIDON-WATCH evolved into a tool for both programmers and medical students to use. We are just starting to make a clean separation between the functionality that is useful for students versus programmers. We plan to develop user profiles that determine the interface behavior in a given situation. The current prototype can only be customized by making changes at the programming level.

GUIDON-WATCH is based on established principles for designing user interfaces on graphical workstations (Ingalls, 1981, Tesler, 1981, Foley and Van Dam, 1982, Card, et al., 1983, Foley, et al., 1984). The design criteria for GUIDON-WATCH emerged from the conventional wisdom on the subject. The user interface is viewed as a conversation consisting of two languages (Foley, et al., 1984): (1) the language in which the user retrieves or requests information (with the mouse), and (2) the program's display and its interpretation. With regard to both languages we aim to maximize expressiveness, understandability, and efficiency. The user should be able to retrieve all information through one interface that is easy to understand and efficient to use. The display should include all relevant information, be easy to interpret, and update quickly when a user makes a request.

Several GUIDON-WATCH users have found the interface simple, consistent, and easy to use. However, those unfamiliar with NEOMYCIN have difficulty realizing exactly how and when the display can be useful. We have found that the display is the best means we have for explaining NEOMYCIN. Therefore, an on-line introduction to GUIDON-WATCH and NEOMYCIN is planned.

Informal evaluation with Stanford University medical students is scheduled for the fall of 1985. Students will *watch* NEOMYCIN diagnose one or more patients. Data records of actual patients will be stored in files that can be accessed by NEOMYCIN during its questioning phase. A student will use GUIDON-WATCH to observe NEOMYCIN's reasoning processes during the consultation. NEOMYCIN will be able to explain in English why it asked a question (Hasling, 1984). Eventually, students will assist NEOMYCIN during a diagnosis in an apprenticeship setting.

The major results to date are summarized below:

- Multiple windows can provide several concurrent views of a knowledge-based system. They help users cope with the complexity of the system by highlighting and summarizing important reasoning events during a problem-solving session.

- Several methods for highlighting facts and events were found effective. These include using different font styles, reverse video, boxing, flashing, and graying regions. Using these techniques, dynamic information associated with a given patient can be overlaid on top of static structures such as a disorder tree or a table of evidence.

- Early results indicate that both programmers and medical users prefer to have GUIDON-WATCH manage screen space automatically. This includes the sizing, placing, and closing of windows. It is not trivial to do this with a large number of windows, particularly during development when changes to the system are frequent. A knowledge-based approach to window management is suggested.

## 3. The Development of NEOMYCIN

The GUIDON project evolved from the MYCIN experiments (Shortliffe, 1976, Buchanan and Shortliffe, 1984) of the 1970s. MYCTN is a rule-based consultation program that recommends drug therapy for certain infectious diseases (e.g., meningitis). Because much of the functionality (e.g., the inference mechanism) of MYCIN does not depend on medical knowledge, it was possible to develop a domain-independent shell called EMYCTN (Van Melle, 1981). MYCIN now consists of EMYCIN plus the MYCIN medical knowledge base. EMYCIN was used to develop several other knowledge-based systems, and is the basis for several commercial products.

In 1979 Clancey completed GUIDON (Clancey, 1983), an intelligent tutoring system that interfaces with EMYCIN. GUIDON can interactively present the rules in an EMYCIN knowledge base to a student. However, Clancey found that the MYCIN rules were often difficult to understand because they combine a diagnostic procedure with medical facts in an opaque manner (Clancey and Letsinger, 1984). In a MYCIN rule the ordering of conjunct clauses in the premise might *implicitly* contain a strategy. For example, a rule might only apply if the patient is an alcoholic. One MYCIN rule premise begins with “if the patient is over 18 years of age and an alcoholic.” The strategy that is implicitly represented in this rule premise is “don’t ask a patient under 18 years of age if they are alcoholic.” As a result, students would not understand many MYCIN rules nor the diagnostic strategy that MYCTN uses

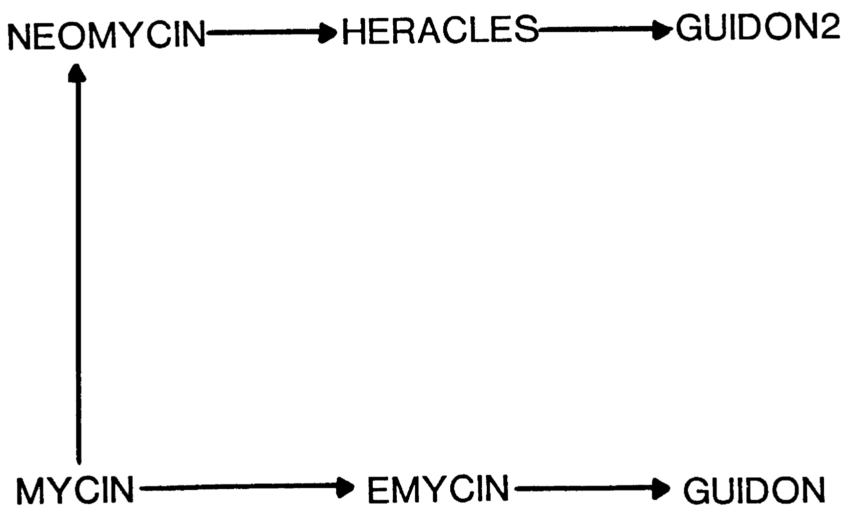


**Figure 3-1:** The evolution of a knowledge-based system

*MYCIN evolved into EMYCIN, a domain-independent shell for building knowledge-based systems. The GUIDON tutoring system is a separate module that could be used with any EMYCIN system. EMYCIN was not found to be an adequate foundation for an instructional program. Therefore, EMYCIN and the MYCIN knowledge base were reconfigured into NEOMYCIN, a medical consultation system which is designed for enhanced explanation and tutoring capabilities. The domain-independent shell that NEOMYCIN is built with is called HERACLES. NEOMYCIN is the basis for GUIDON2, a tutoring system now in development.*

implicitly.

GUIDON demonstrated that satisfying the requirements for expert performance are not necessarily sufficient for the purpose of explanation and tutoring. Therefore, MYCTN was significantly reconfigured into a new program called NEOMYCIN (Clancey and Letsinger, 1984) that represents a diagnostic strategy separately from medical facts. For example, the diagnostic strategy used in NEOMYCIN explicitly states to check for conditions that would make a question inappropriate. The knowledge base has also been expanded to include diseases that can be confused with meningitis (this is important for instruction). NEOMYCIN is the foundation for GUIDON-2, a new series of instructional programs. GUIDON-WATCH is the first component of the GUIDON-2 system. Importantly, interactive graphics makes knowledge and reasoning visible only to the extent that the knowledge is represented *explicitly* in a program. The well-structured and explicit design of NEOMYCIN provides many opportunities for exposing the program's reasoning to students and other users.

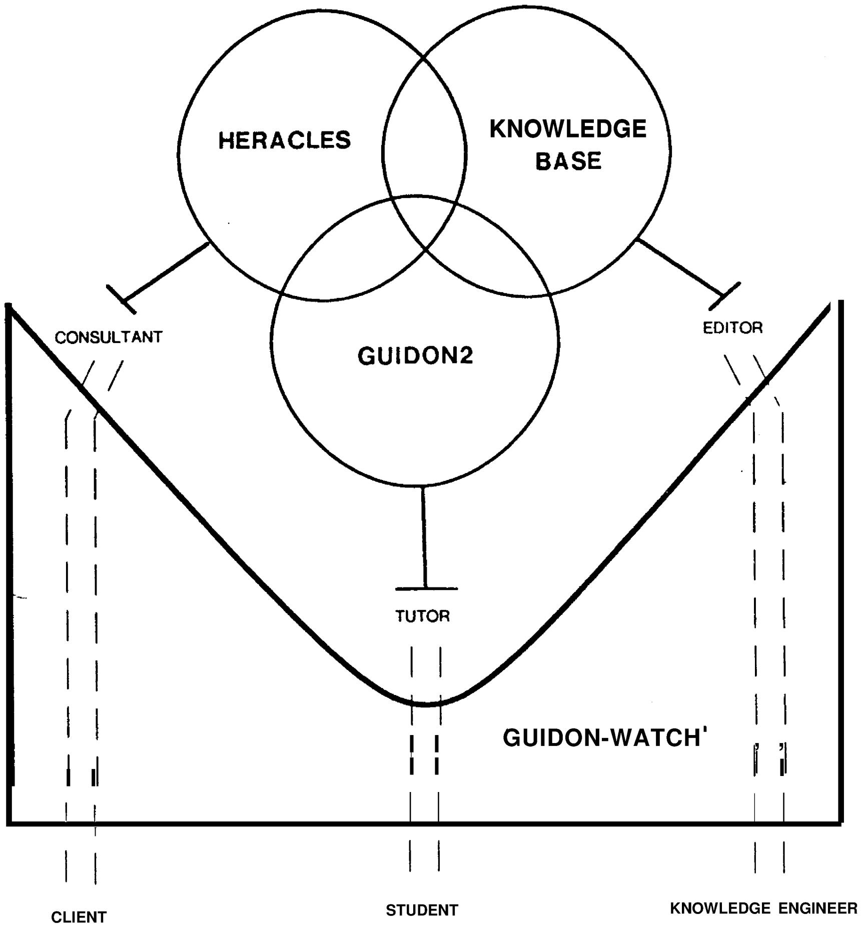


**Figure 3-2:** The HERACLES architecture.

*The relationship between GUIDON-WATCH and three primary system modules is illustrated above. A consultation system consists of HERACLES, a knowledge base, and GUIDON-WATCH. For instructional use, the GUIDON2 module (now in development) can be added. GUIDON-WATCH provides an interface for instructional use, to run consultations, and to edit the knowledge base. Although there are differences in the interface for each kind of user, in general, the interface is very similar and represents a single program with several modes of behavior. (The graphic editor is not described in this paper because that interface has not been completey integrated with HERACLES.)*

NEOMYCIN has led to a domain-independent system called HERACLES. HERACLES is to NEOMYCIN as EMYCIN is to MYCIN (Figure 3-l). In other words, NEOMYCIN consists of HERACLES and a medical knowledge base (Figure 3-2). HERACLES is a software tool applicable to diagnostic problems in many domains. For example, HERACLES was used to develop a knowledge base for cast iron fault diagnosis (Thompson and Clancey, 1985). The HERACLES program includes a diagnostic procedure represented in a rule-based declarative language, rule interpreters, a set of domain relations (e.g., causes, subtype, suggests), various software tools for developing knowledge-based systems (many derived from EMYCIN), an explanation facility, and GUIDON-WATCH. To construct a specific consultation program, the system designer adds a knowledge base of facts and rules. All the examples in this paper use the NEOMYCIN knowledge base, but GUIDON-WATCH will work with any HERACLES knowledge base.

## 4. Description of the GUIDON-WATCH display

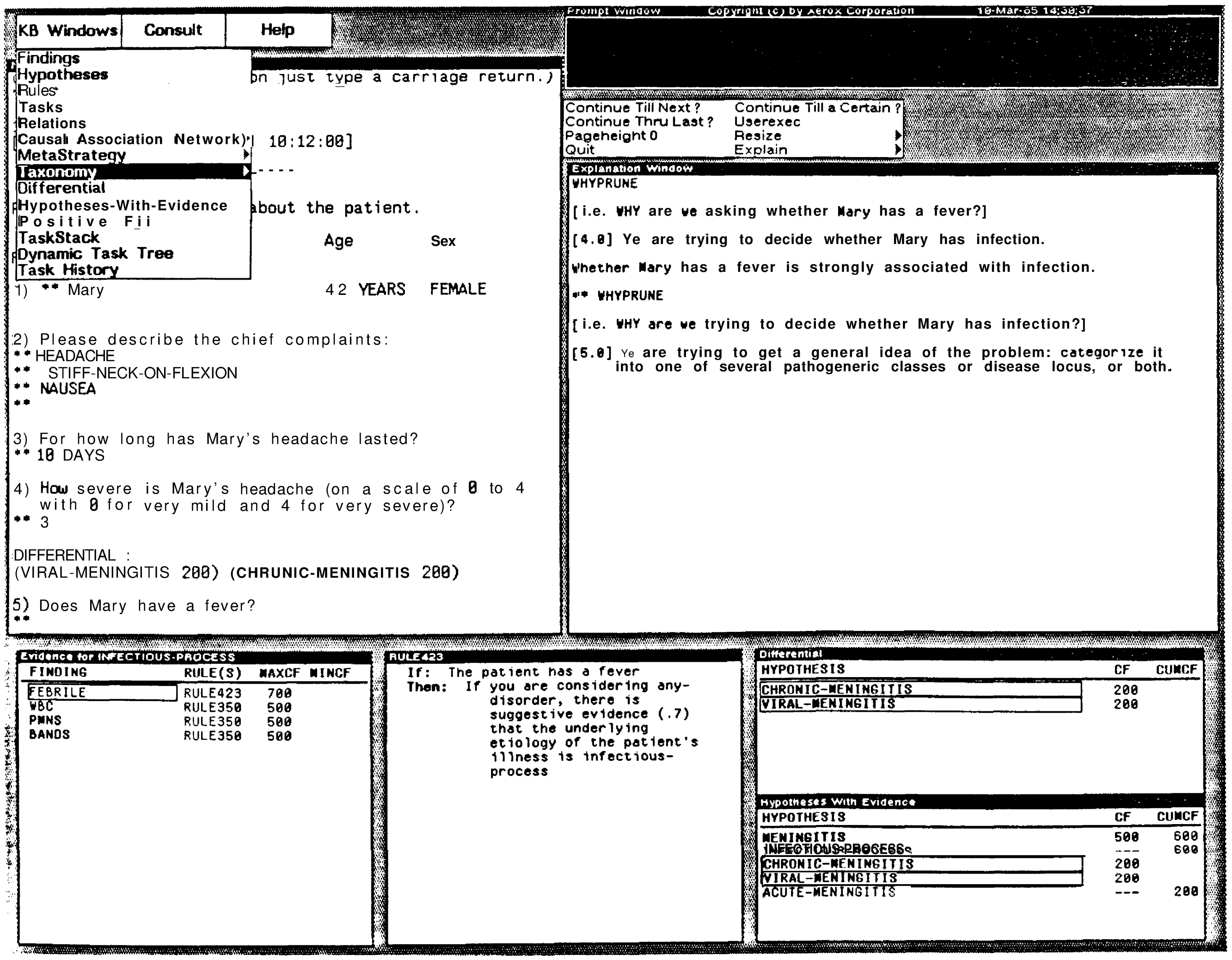
The windows and menus used in GUIDON-WATCH are described in detail in this section. First, the programming environment is briefly described to show what tools we started with. Second, an overview of the interface is provided. The third section describes the display facilities in detail.

### 4.1. Programming Environment

*GUIDON-WATCH is implemented on XEROX 1100 Series workstations running INTERLISP-D. The black and white display screen is 1024 pixels wide by 808 pixels high, which provides approximately 75 pixels per inch resolution. INTERLISP-D provides a window package that supports multiple overlapping windows, scroll bars, and other window operations (Sannella, 1983). Many graphics primitives are provided for drawing lines and curves, manipulating bitmaps, filling and manipulating regions, checking the state and position of the mouse, etc. In addition, a menu package, a grapher package (e.g., to display trees), and several default window functions (e.g., scrolling by repainting) are provided. It required only a page of code to implement a simple pull-down menu package using the window primitives.

### 4.2. Overview of the User Interface

Pull-down menus and a Prompt Window are at the top of the GUIDON-WATCH display. (Figure 4-l). The Prompt Window is a standard part of the INTERLISP-D user interface and is used to print messages. Currently there are three pull-down menus of interest to a medical student: ***KB*** *(Knowledge Base) Windows, Consult,* and *Help.* The ***KB*** *Windows* pull-down menu displays a list of windows that can be opened for browsing the knowledge base or viewing a consultation. The *Consult* menu is used to start and quit a consultation. The *Help* menu allows the user to obtain information on the contents of a window.



```
on just type a carriage return.)
                    10:12:00]
                    -----
                    about the patient.
                                  Age        Sex
1) ** Mary                        42 YEARS   FEMALE

2) Please describe the chief complaints:
** HEADACHE
**   STIFF-NECK-ON-FLEXION
** NAUSEA
**

3) For how long has Mary's headache lasted?
** 10 DAYS

4) How severe is Mary's headache (on a scale of 0 to 4
   with 0 for very mild and 4 for very severe)?
** 3

DIFFERENTIAL :
(VIRAL-MENINGITIS 200) (CHRUNIC-MENINGITIS 200)

5) Does Mary have a fever?
**
```

```
WHYPRUNE

[i.e. WHY are we asking whether Mary has a fever?]

[4.0] We are trying to decide whether Mary has infection.

Whether Mary has a fever is strongly associated with infection.

** WHYPRUNE

[i.e. WHY are we trying to decide whether Mary has infection?]

[5.0] We are trying to get a general idea of the problem: categorize it
      into one of several pathogeneric classes or disease locus, or both.
```

| FINDING | RULE(S) | MAXCF | MINCF |
|---|---|---|---|
| FEBRILE | RULE423 | 700 | |
| WBC | RULE350 | 500 | |
| PMNS | RULE350 | 500 | |
| BANDS | RULE350 | 500 | |

```
If:    The patient has a fever
Then:  If you are considering any-
          disorder, there is
          suggestive evidence (.7)
          that the underlying
          etiology of the patient's
          illness is infectious-
          process
```

| HYPOTHESIS | CF | CUMCF |
|---|---|---|
| CHRONIC-MENINGITIS | 200 | |
| VIRAL-MENINGITIS | 200 | |

| HYPOTHESIS | CF | CUMCF |
|---|---|---|
| MENINGITIS | 500 | 600 |
| INFECTIOUS-PROCESS | --- | 600 |
| CHRONIC-MENINGITIS | 200 | |
| VIRAL-MENINGITIS | 200 | |
| ACUTE-MENINGITIS | --- | 200 |

**Figure 4-1:** A GUIDON-WATCH display during a consultation.

*The user is running a consultation and the system has paused at question 5. The user has opened several windows to get information about the hypotheses that are being considered at this time. The use of pull down menus is also illustrated: the user has selected the* **KB** *Windows menu and moved the mouse over the menu item Taxonomy.* ***If*** *the user releases the mouse button now, the Taxonomy Window will be displayed.*

### 4.2.1. Use of the mouse

XEROX 1100 computers can be used with either a two or three button mouse (selecting the left and right button at the same time on a two button mouse is equivalent to pressing the middle button on the three button mouse). In GUIDON-WATCH, the mouse is used in a simple and consistent way. The left button is used to select all menu items and text items in a window. For example, in a window that displays a list of diseases, the user can select the name of a disease using the left button. A pop-up menu is displayed that allows a user to get more information in another window (Figure 4-2). Only those items that are currently relevant appear in the pop-up menu.

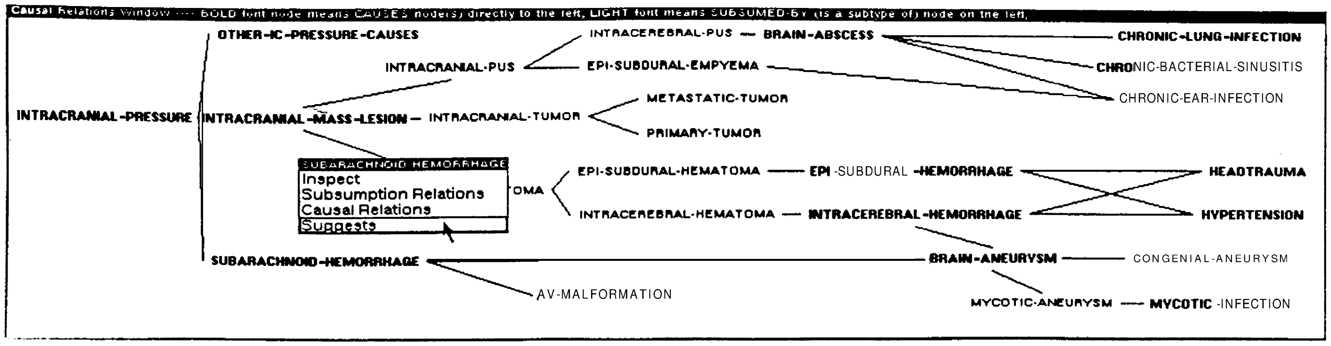

Causal Relations Window ---- BOLD font node means CAUSES node(s) directly to the left, LIGHT font means SUBSUMED-BY (is a subtype of) node on the left.

**Figure 4-2:** The Causal Relations Window.

*The user has selected the node SUBARACHNOID-HEMORRHAGE with the left button and a pop-up menu is displayed with items for displaying additional information. This graph was automatically generated from the NEOMYCIN knowledge base, edited by hand to fit on the screen, and then stored on a file. If the user wished to display the graph with a different root node GUIDON-WATCH dynamically generates the graph at* ***runtime.***

It has not been decided whether or not students will be asked to use more than the left button. In our current programming environment, the right button is used in the default manner provided by INTERLISP-D, to manipulate windows (e.g., reshaping, closing). The middle button is sometimes used to display a pop up menu with items that apply to the entire data structure in a window. For example, a user may want to highlight those items in a window that all have a certain property. We are considering the use of icons in a window for operations besides selecting menu or text items (e.g., closing a window). Therefore, the student interface may only use one button.

### 4.2.2. On-line Help

If the user selects *Help window* from the *HELP* pull-down menu, then a **HELP** icon attaches to the mouse cursor. The user can get help about a window by moving the **HELP** icon into a window and buttoning the window. A message associated with the selected window is printed in a special help window.

### 4.2.3. Management of windows

The **INTERLISP-D** graphics package provides functions for prompting users to position a ghost image of a window or to a shape a window. These prompts can be confusing to novices and distract from the task at hand. If it is possible to make a good decision regarding the size and position of a window, then we can free the user from this chore. In addition, an automatic window management system can often optimize the use of screen space better than a user. This is true **in** GUIDON-WATCH because there are a known set of windows whose contents are constrained to a certain form (e.g., a table).

To manage the window display, the screen is divided into logical units. The GUTDON-WATCH screen consists of a top, middle, and bottom section. The top section contains the pull-down menus and the Prompt Window. The bottom and middle sections display knowledge base structures and have well-defined lower borders. Another logical division of the **GUTDON-**WATCH display provides vertical boundaries. For example, the width of the screen can be divided into equal or unequal regions. The current prototype uses three regions with two equal and one slightly wider than the other two. Furthermore, you can have a hierarchy of subdivisions (i.e., regions) Each window in GUIDON-WATCH is associated with one or more regions where it can be displayed.

GUIDON-WATCH decides where to place a window based on several considerations: (1) the default region of the window, (2) the other windows that are displayed and their position, and (3) the set of windows that the user would mostly likely prefer to remain in view. While the current window management system is effective, we would like to extend the flexibility of the interface. This would require a more complex scheme. It might be necessary to consider moving or reshaping windows that are already on the screen. Note that window systems that provide this capability do not consider the *semantics* of the contents of windows. Therefore, algorithms for scaling pictures and changing the font size of text are not sufficient when you have to decide where windows should be placed and which windows should be closed or covered.

Although flexibility and control are relinquished by the user, the benefits of automatic screen management seem to outweigh potential disadvantages. Automatic window management saves the user time and maximizes the use of screen space. It is possible to allow the user to turn off automatic features, change defaults, or allow the user to use the move and reshape facilities. Furthermore, menus or icons can be used to allow the user to choose from a predefined set of sizes, positions, fonts, etc., but then the implementation of the automatic window manager becomes increasingly complex. In our current implementation, when a window is displayed, a complex conditional in the window's display function is evaluated. This code is difficult to understand and modify. In addition, the situation has been complicated by the need for different user profiles. We are considering an approach where the behavior of the interface is specified separately and declaratively using knowledge representation formalisms (e.g.. rules) and object-oriented programming.

### 4.2.4. Dynamic updating of the screen display

Displaying dynamically changing information presents problems that are not unique to our application. For example, how often do you update the screen? Do you gray out regions that are out of date or immediately update them? Our philosophy is that users should be able to open and close windows at any time and that the display should accurately reflect the current state of the system or gray out regions that are not continuously updated. Regions that are grayed out can either be automatically updated at specified intervals or manually updated by having the user simply button the window to redisplay itself.

## 4.3. The GUIDON-WATCH windows

This section describes many of the windows available to the GUIDON-WATCH user and addresses the following important issues: What information in a HERACLES knowledge base is most important to display? For programmers? For medical students? How can dynamic information be displayed? In the first subsection below, the focus is on static knowledge structures and how they are displayed in GUIDON-WATCH. Subsequent subsections discuss the display of dynamic consultation knowledge.

### 4.3.1. What is there to display in a knowledge base?

A HERACLES knowledge base (e.g., NEOMYCIN medical knowledge base) includes findings, hypotheses, rules, tasks, and relations among theses. Findings are data that can be requested or inferred from rules. Generally findings can be observed or measured. Hypotheses can only be inferred from rules. In NEOMYCIN, hypotheses include diseases and pathophysiological states. Relations refer to predicate calculus relations and in HERACLES include *subtype, causes,* etc. *Static* knowledge includes facts about findings and hypotheses as defined by relations (e.g., meningitis is a subtype of infection, headache is a finding, etc.). It also includes the diagnostic procedure and domain rules (e.g., if the patient has double vision, then there is suggestive evidence for intracranial pressure). *Dynamic* knowledge is situation specific and refers to information that becomes known only during a problem solving session (e.g., "Mary's temperature is 102 degrees.").

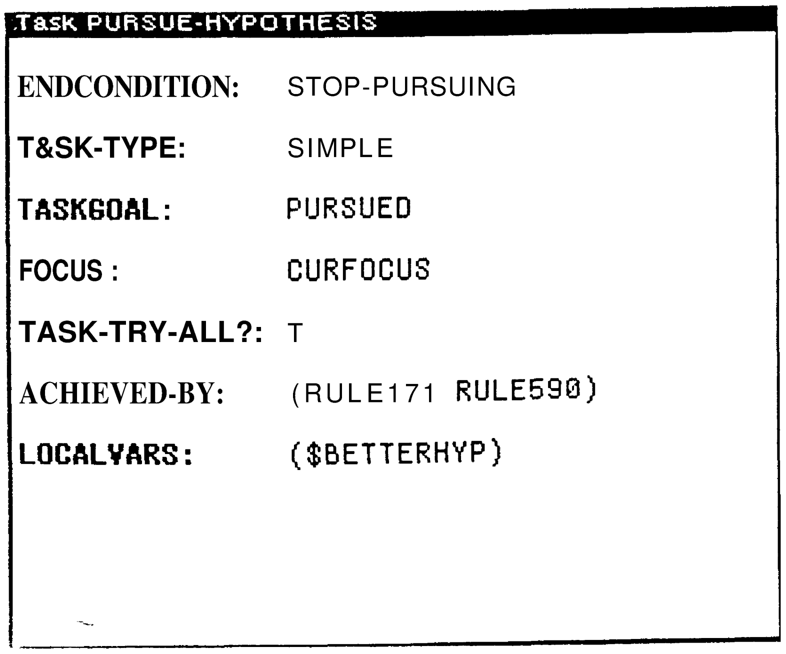
NEOMYCIN uses a diagnostic strategy known as heuristic classification (Clancey and Letsinger, 1984). Given an enumerated set of solutions (e.g., diseases or possible diagnoses), NEOMYCIN heuristically maps a set of findings onto one or more possible solutions. This diagnostic procedure is provided by HERACLES (Heuristic Classification Shell). It is represented as tasks, which are procedures that are stated in a declarative rule-based language (Figure 4-3). When a task is invoked, one or more of its metarules are applied (Figures 4-4, 4-5). Metarules in HERACLES are similar to conditionals in a procedure, but they are expressed as abstract rules.

Windows that display static knowledge include the task and metarule windows in Figures 4-3, 4-4, and 4-5. They also include the Findings, Hypotheses, and Relations windows, which simply display an alphabetical ordering. Other windows display a graph to show the relationships between groups of objects. The Causal Relations Window (Figure 4-2) is a lattice with causal and subtype links between findings and hypotheses; the Diagnostic Strategy window (Figure 4-6) shows the calling structure of the diagnostic tasks; and the Taxonomy window (Figure 4-7) represents a subtype hierarchy of disorders. In all of these windows the user may select an item to get more information.

### 4.3.2. Reifying the process

The windows described below all display dynamic information during a consultation.

*The Taxonomy and Causal Relations Windows.* An important concept in medical diagnosis is the *differential,* the set of competing hypotheses currently being considered. The etiological


Task PURSUE-HYPOTHESIS

| | |
|---|---|
| ENDCONDITION: | STOP-PURSUING |
| T&SK-TYPE: | SIMPLE |
| TASKGOAL: | PURSUED |
| FOCUS : | CURFOCUS |
| TASK-TRY-ALL?: | T |
| ACHIEVED-BY: | (RULE171 RULE590) |
| LOCALVARS: | ($BETTERHYP) |

**Figure 4-3:** The Task Property Window.

*Above the properties and values of the task Pursue-Hypothesis are displayed.*

taxonomy **is** a tree of possible diagnoses **or** solutions in NEOMYCIN. The differential represents a *cut* through this solution space. Boxing the hypotheses in the Taxonomy Window that are on the differential is a simple way to make this *cut* visible (Figure 4-7).

Flashing and boxing nodes in the Taxonomy and Causal Relations Windows emphasize the dynamic search strategy. Whenever a hypothesis is added to the differential, its corresponding node label is flashed and then boxed. Whenever the hypothesis is removed from the differential, the box is redrawn with lighter lines, so that the hypotheses that had been considered previously are still highlighted, but the ones currently on the differential are more prominent (Figure 4-7). **A** student can observe NEOMYCIN *looking up* the disorder tree to group and compare categories of disorders before *looking down* to refine hypotheses.

Conclusions in a HERACLES consultation are associated with *certainty factors* that represent a degree of belief. They are not probabilities. In HERACLES, each hypothesis has both a certainty factor (CF) and a cumulative certainty factor (CUMCF). The CF represents the

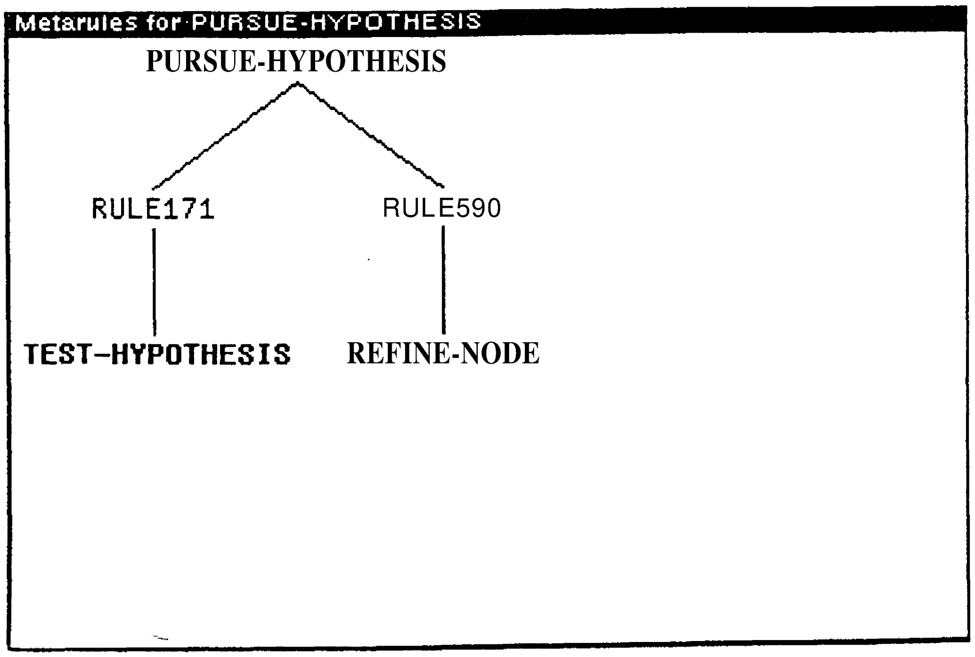
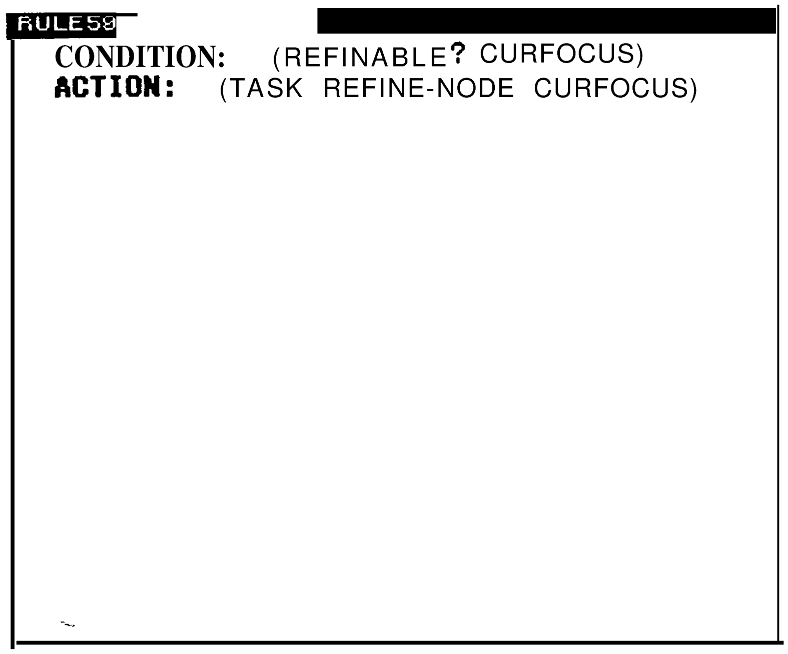


**Figure 4-4:** The Metarules Window.

*Above the metarules that the task Pursue-Hypothesis calls are displayed.*

combined certainty of all rules that have concluded directly about the hypothesis. The CUMCF represents a combination of the CF of a given hypothesis (which may be zero) with **CFs** of its descendants in the disorder taxonomy. For example, evidence for meningitis (a positive CF) increases the CUMCF of infectious process because meningitis is a subtype of infectious process. (To be exact, negative **CFs** of ancestors are also combined; therefore, evidence against infection can decrease the CUMCF of meningitis.)

When the CF or CUMCF is updated for a hypothesis, new values are printed below the node label corresponding to the hypothesis. The CF is printed on the left, the CUMCF on the right if it differs. The figures in this paper show the **CFs** printed as integers from -1000 to **+1000;** this is how they are represented internally. These numbers are far from precise and should be interpreted as falling in several categories: definite, strongly suggestive, suggestive, weakly suggestive or no evidence for (or against) a hypothesis. For students the internal CF values will not be printed; instead a graphic notation, such as zero to four pluses or minuses could be



RULE59
CONDITION: (REFINABLE? CURFOCUS)
ACTION: (TASK REFINE-NODE CURFOCUS)

**Figure 4-5:** The Rule Window displaying a metarule of the task Pursue-Hypothesis.

used to indicate the degree of belief.

In the case in which a hypothesis window is not open, the printing, boxing and flashing of nodes is not done immediately. However, whenever the Taxonomy or Causal Relations window is opened during a consultation, the window is updated so that all the hypotheses are appropriately boxed, and certainty factors are printed. Therefore, the user is free to open these windows at any time. Below we describe several other windows that display dynamic information.

*The Consultation Typescript.* The Consult window (Figure 4-l) is opened when a user starts a consultation using the *Consult* menu. This window displays the questions that are asked during a consultation. Each question is followed by a response that is either supplied by the user or is retrieved automatically from a patient data file. Before an answer is retrieved the program pauses. The user can then use the mouse to select items or open any windows. A menu is provided that allows a user to proceed one or more questions further, receive textual explanations, resize the consultation window, etc.

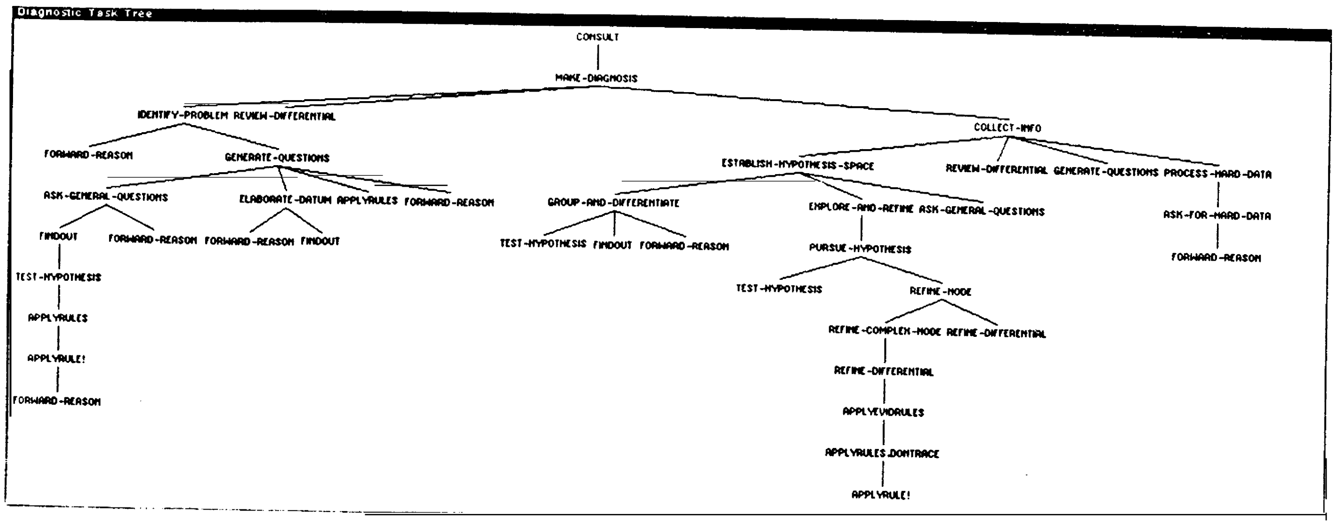


**Figure 4-6:** The Diagnostic Task Tree Window.

*When a task is selected in this window, a menu pops up that allows the user to display either the properties of the task in the Task Property window (Figure 4-3) or the metarules that a task applies in the Metarules window (Figure 4-4). During a consultation the user can also choose to see dynamic information about task calls. This is described in the section Dynamic Task Windows.*

*Evidence Window.* This window can be displayed without running a consultation. However, during a consultation dynamic information is overlaid onto static knowledge structures to show the current evidence relations between findings and hypotheses (Figure 4-8). All potential evidence for a hypothesis is displayed as a table in this window. The first column lists findings and hypotheses that suggest a hypothesis. The second column lists the rules that use these findings or hypotheses to conclude about the hypothesis. The third column shows the maximum CF in the rule's action, and the fourth column shows, if different, the minimum CF in the rule's action.

During a consultation, additional information is provided to the user through the use of bold fonts and graying over regions. A finding with a positive value is printed in bold; a negative finding is grayed over. Analogously, rules that have succeeded are printed in bold; rules that have failed are grayed over. Findings and rules that appear in normal print have not been investigated yet. This simple notation is an effective means of providing a great deal of

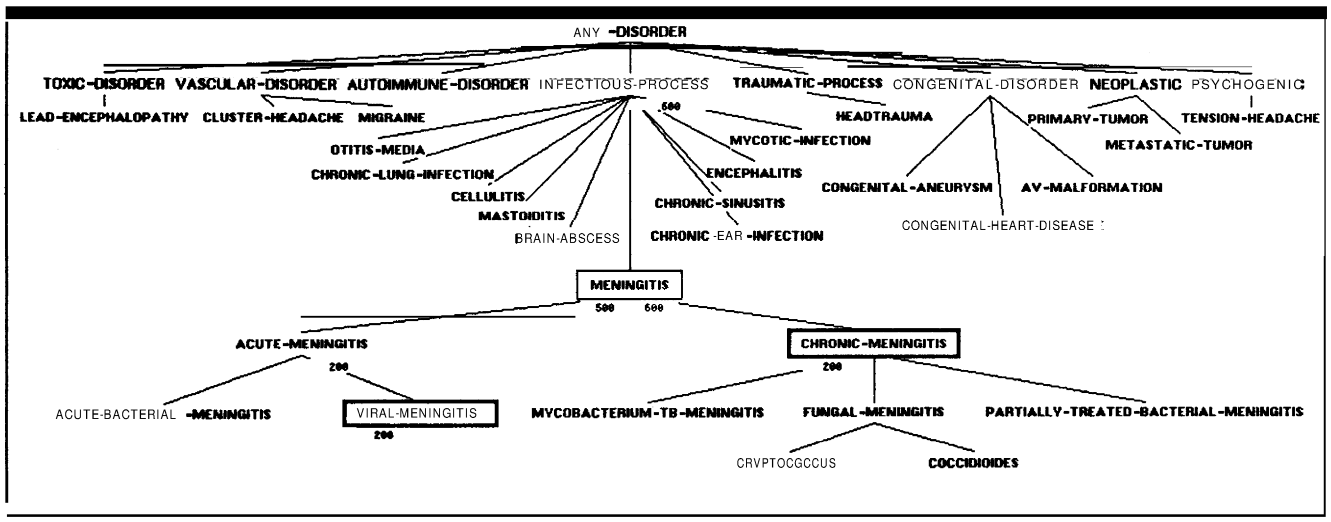


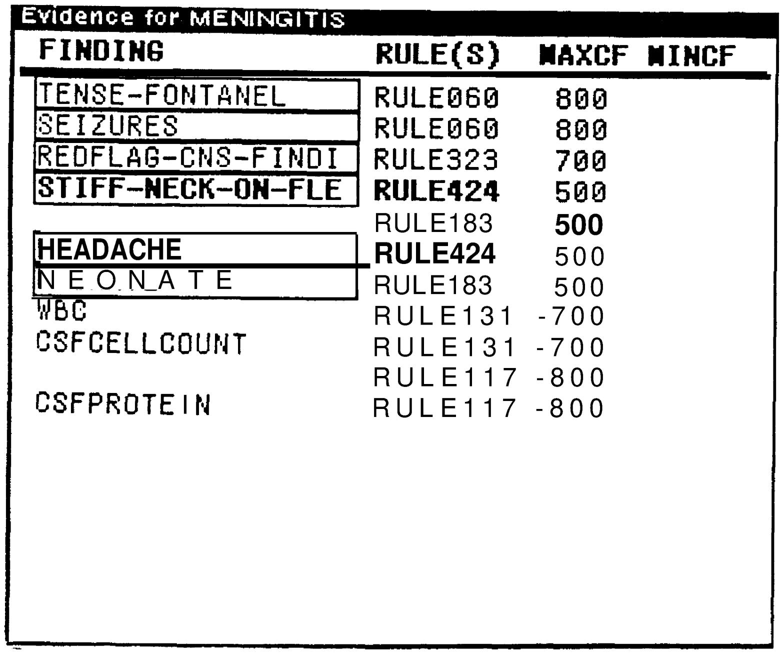
**Figure 4-7:** The Taxonomy Window.

*The boxing, flashing, and printing techniques used in this window make the dynamic search strategy visible by displaying dynamic information on top of a static knowledge structure, in this case the etiological taxonomy. The differential shown here is NEOMYCIN's internal differential and may not correspond precisely to a physician's differential. The differential shown to a student may differ from NEOMYCIN's internal list. This graph was generated, edited, and stored in the same manner as the graph in Figure 4-2. Note that Figures 4-7 through 4-10 correspond to the same state of the consultation as displayed in Figure 4-1.*

information in a concise and understandable manner. Furthermore, it illustrates how dynamic information can be displayed on top of static knowledge structures that are displayed in a table format.

*Positive Findings Window.* The Positive Findings Window (Figure 4-10) displays all the findings that have a positive value (i.e., the value is "yes", a number, or symbolic). Findings are printed in the first column, values in the second column, and CFs in the third (printed only if less than 1000). Findings are selectable, and when buttoned a pop-up menu is displayed. For example, a user may want to select a finding to find out which hypotheses the finding may suggest.

In this window items are printed incrementally during a consultation. If the Positive Findings window is open, then new positive findings are printed in the window as soon as they



Evidence for MENINGITIS

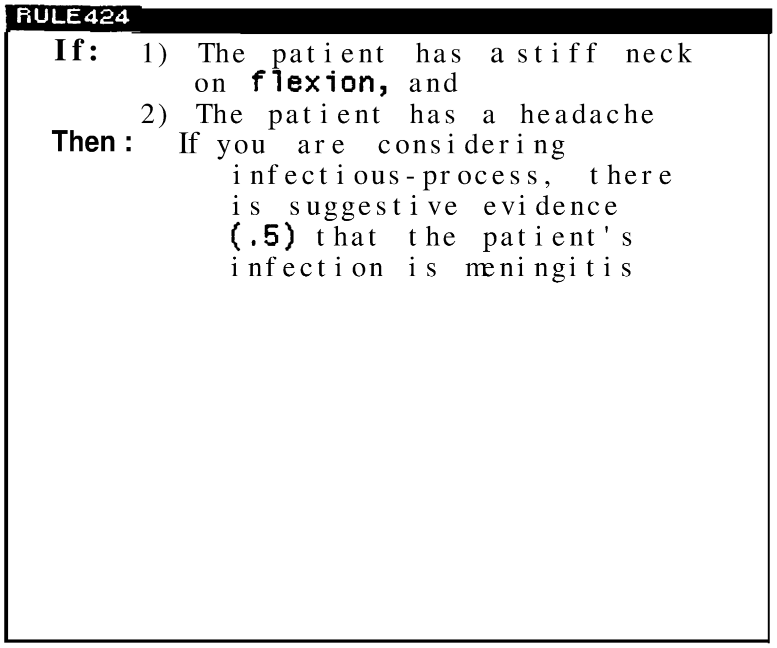
| FINDING | RULE(S) | MAXCF | MINCF |
|---|---|---|---|
| TENSE-FONTANEL | RULE060 | 800 | |
| SEIZURES | RULE060 | 800 | |
| REDFLAG-CNS-FINDI | RULE323 | 700 | |
| **STIFF-NECK-ON-FLE** | **RULE424** | 500 | |
| | RULE183 | **500** | |
| HEADACHE | **RULE424** | 500 | |
| NEONATE | RULE183 | 500 | |
| WBC | RULE131 | -700 | |
| CSFCELLCOUNT | RULE131 | -700 | |
| | RULE117 | -800 | |
| CSFPROTEIN | RULE117 | -800 | |

Figure 4-8: The Evidence Window.

*The findings and hypotheses displayed in this window are ordered so that the ones that may be most suggestive (have the highest* ***MAXCF)*** *are on top. The user may have selected Meningitis in the Taxonomy Window because it is on the differential with a positive CF, and then displayed this Evidence Window. The user may then display a rule that succeeded or failed in the Rule Window. For example, RULE424 is printed in bold in the Evidence Window above showing that the rule succeeded. This rule is then displayed in the Rule Window (Figure 4-P).*

are known. If the window is closed, then the whole list of positive findings is printed when the window is opened. This feature provides flexibility for the user, who can open or close the window at any time during a consultation.

*Differential and Hypotheses With Evidence Windows.* Hypotheses on the differential are boxed when they appear in certain windows. However, the differential is such an important structure that a special window is provided for its display (Figure 4-11). However, not all hypotheses for which there is positive evidence are on the differential at a given time. Another important class of hypotheses are all those for which there is belief. This includes hypotheses for which there is direct evidence (i.e., at least one rule concluded the hypothesis) and those for which there is belief when propagation is included (i.e., the CUMCF is above a certain



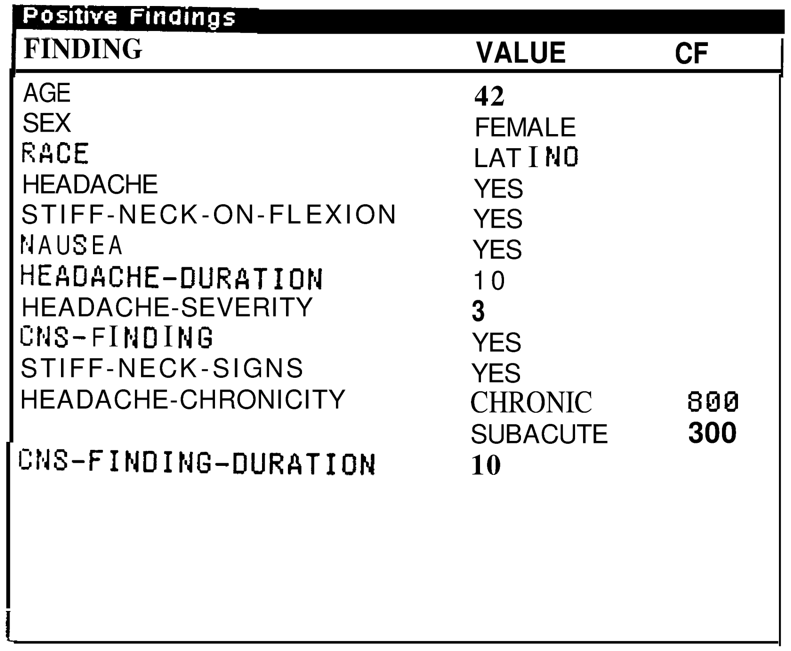
```
RULE424
If:   1) The patient has a stiff neck
         on flexion, and
      2) The patient has a headache
Then:  If you are considering
          infectious-process, there
          is suggestive evidence
          (.5) that the patient's
          infection is meningitis
```

**Figure 4-9:** The Rule Window displaying a domain rule.

threshold). These are displayed in a separate window (Figure 4-12).

In both the Differential and Hypotheses-With-Evidence windows, hypotheses that have direct evidence supporting them are printed in bold. These windows also contain columns for CF and CUMCF values. As usual, the hypotheses are selectable. These two windows, as well as the -Taxonomy and the Causal relations windows, illustrate how GUIDON-WATCH provides *multiple views* of the same knowledge structures.

*Dynamic Task Windows.* These windows provide users with dynamic views of the diagnostic strategy as it is instantiated during a consultation. This is a challenging presentation problem because the abstract nature of the diagnostic procedure as it is represented in the task and metarules is not nearly as intuitive to people as are disorder hierarchies, causal networks, domain rules, and lists of findings. Although the goal is to provide a view of NEOMYCIN's reasoning that is understandable to medical students, the model of the diagnostic strategy is in the form of a complex procedure that is intimately tied to basic concepts of computing. For example, task calls are very similar to procedure calls; a task may have a focus and local variables. A focus consists of one or more findings, hypotheses, or rules depending on the task.


Positive Findings

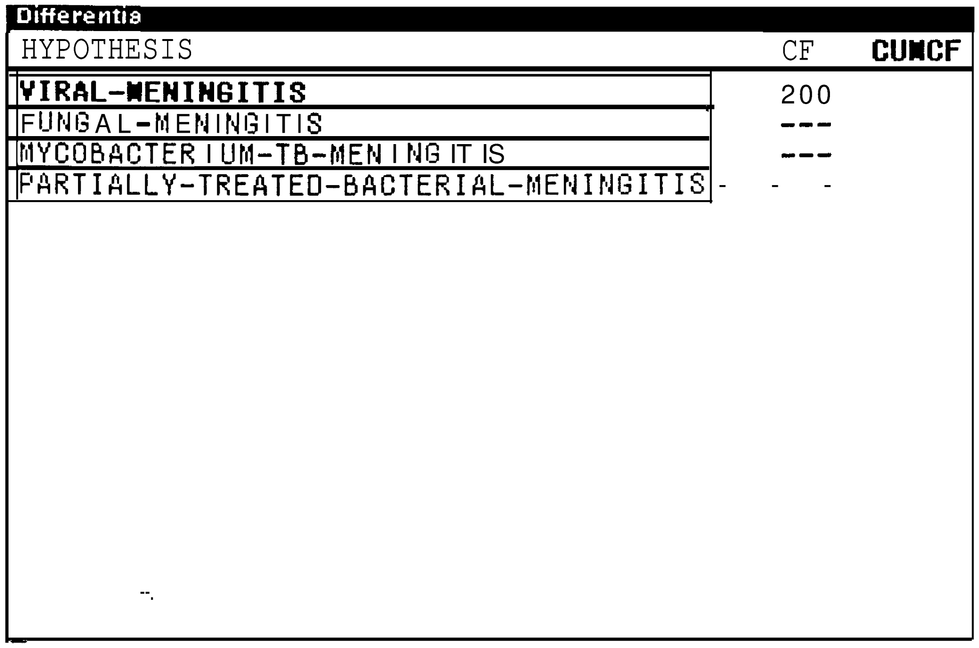
| FINDING | VALUE | CF |
|---|---|---|
| AGE | 42 | |
| SEX | FEMALE | |
| RACE | LATINO | |
| HEADACHE | YES | |
| STIFF-NECK-ON-FLEXION | YES | |
| NAUSEA | YES | |
| HEADACHE-DURATION | 10 | |
| HEADACHE-SEVERITY | 3 | |
| CNS-FINDING | YES | |
| STIFF-NECK-SIGNS | YES | |
| HEADACHE-CHRONICITY | CHRONIC | 800 |
| | SUBACUTE | 300 |
| CNS-FINDING-DURATION | 10 | |

**Figure 4- 10:** The Positive Findings Window.

For example, Test-Hypothesis may have meningitis as a focus in NEOMYCIN. Tasks invoke other tasks in a chain, similar to procedure calls.

The three windows described below each provide a different view of the dynamic diagnostic strategy by using three different graphic formats: a stack, a tree, and a table. Although it has not yet been decided how they will be adapted for instruction, they are already very useful for programmers trying to debug or understand NEOMYCIN's behavior. Programmers can use these windows to find out exactly what NEOMYCIN is doing or has done at a detailed strategic level. Consistent with Model's (Model, 1979) recommendations, these windows provide monitoring and debugging tools at a level that corresponds to the program's design (e.g., tasks and metarules). This is a great improvement over examining the low-level LISP stack which reflects the strategy only in a very indirect way.

*Task Stack Window.* This window displays the current stack of task calls, which is similar to a stack of procedure calls (Figure 4-13). The task calls are printed from in the order that they were called, with the first task printed at the top of the window. If the task has a focus, then it is printed in square brackets after the task. The metarule that the task successfully applied is



Differentia

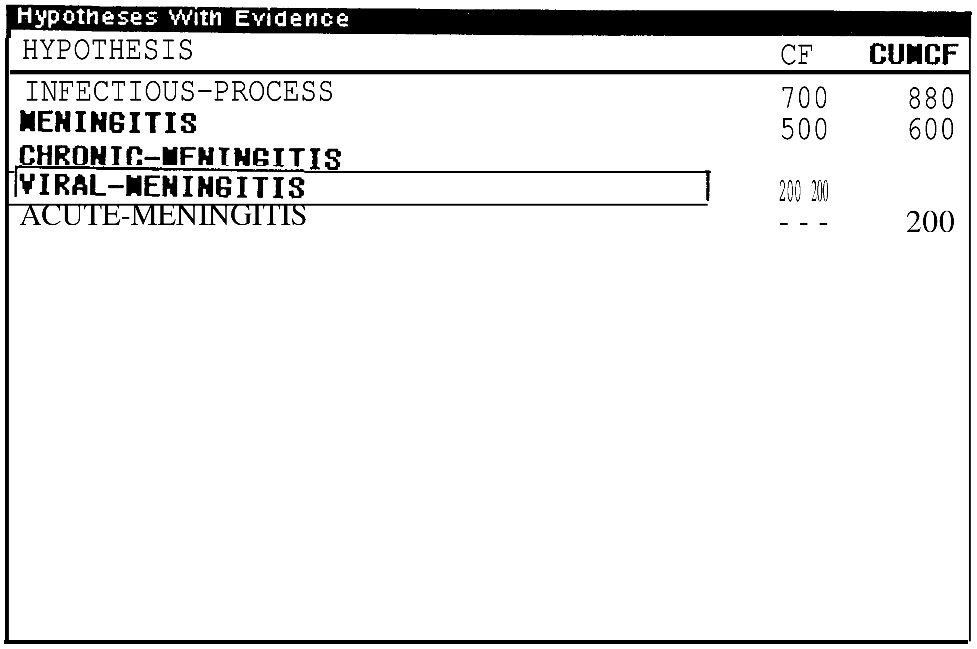
| HYPOTHESIS | CF | CUMCF |
|---|---|---|
| VIRAL-MENINGITIS | 200 | |
| FUNGAL-MENINGITIS | --- | |
| MYCOBACTERIUM-TB-MENINGITIS | --- | |
| PARTIALLY-TREATED-BACTERIAL-MENINGITIS | - - - | |

Figure 4-11: The Differential Window.

*This is the differential several questions after the point shown in Figure 4-1. Subsequent figures all show windows as displayed at this later point.*

printed below the task. Metarules are *attached* to the task they invoke by a vertical line. ‘Different font faces are used to distinguish tasks, metarules, and foci from one another. Every rule, finding, and hypothesis in the Task Stack Window is selectable so that the user can quickly get more detailed information on an item of interest.

The Task Stack Window provides a view of the current path through the diagnostic tree with metarules and foci instantiated. By examining the task stack, the user can understand the reason for the current strategy. For example, the user may be interested in why a question is being asked. Students will be able to get textual explanations that should satisfy their needs (Figure **4-1**), but programmers may want to examine the task stack to understand the computational reasons for a data request at the task and metarule level. By examining the task stack in Figure (4-13) the user can see that NEOMYCIN is testing the hypothesis Mycobacterium-TB-Meningitis. As a result, a rule that was applied led to a series of calls to



Hypotheses With Evidence

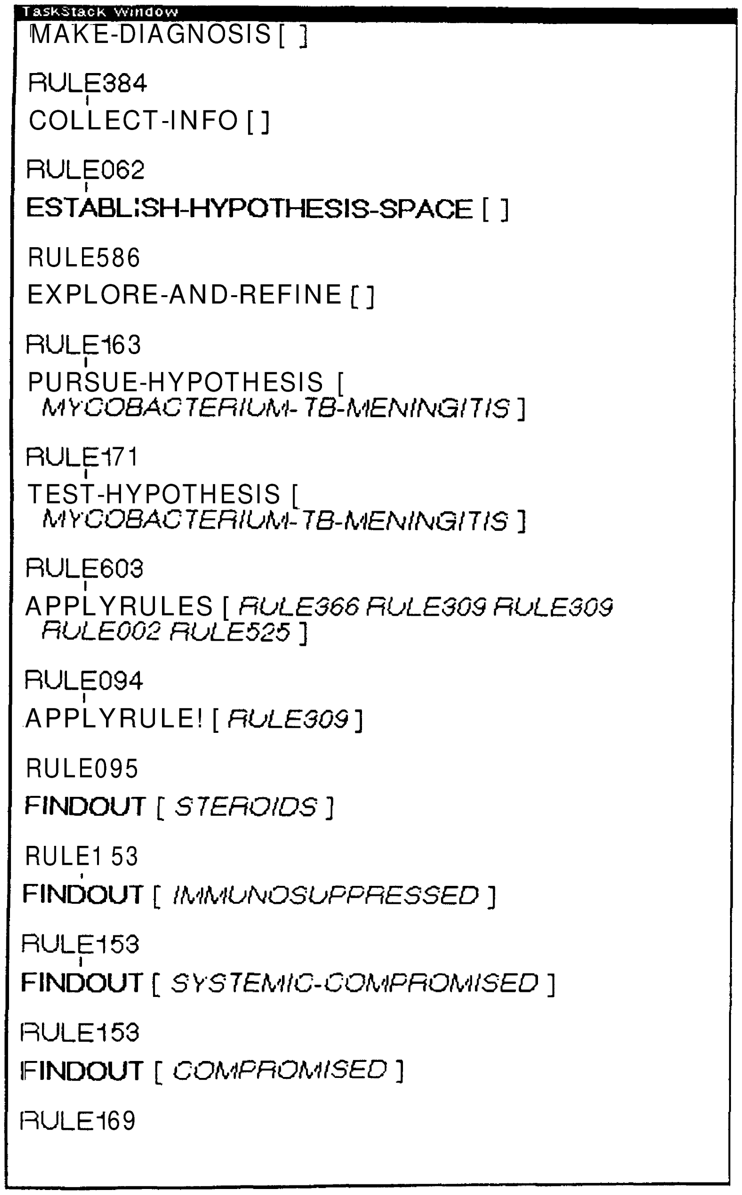
| HYPOTHESIS | CF | CUMCF |
|---|---|---|
| INFECTIOUS-PROCESS | 700 | 880 |
| **MENINGITIS** | 500 | 600 |
| **CHRONIC-MENINGITIS** | | |
| **VIRAL-MENINGITIS** | 200 200 | |
| ACUTE-MENINGITIS | --- | 200 |

Figure 4-12: The Hypotheses With Evidence Window.

the task **Findout.** The last call, with the focus “compromised,” finally resulted in the question “Is Mary a compromised-host?,” which the user would see in the Consultation Typescript Window.

*Dynamic Task Tree Window.* This window displays a graph that shows all or part of the dynamic history of task calls (Figure 4-14). This allows a user to view the overall *structure* of the diagnostic strategy that NEOMYCIN is using during or at the end of a consultation. This is useful because the static Diagnostic Task Tree Window (Figure 4-6) shows all possible paths in the task tree; this window shows only the paths that are part of an actual diagnosis and reveals patterns of multiple calls of the same task.

*Task History Window.* This window contains a table of all the invocations of any given task during the consultation (Figure 4-15). More or less, it provides an alternate view (for a given task) of the information displayed in the Dynamic Task Tree Window. In the first column, the invocation number of the task is printed, with “1” meaning the first time the task was called. In the second column, the focus of the task call is printed; in the third column, the metarule that invoked the task is printed; and in the fourth column, the calling task is printed.


TaskStack Window

```
MAKE-DIAGNOSIS [ ]
RULE384
COLLECT-INFO [ ]
RULE062
ESTABLISH-HYPOTHESIS-SPACE [ ]
RULE586
EXPLORE-AND-REFINE [ ]
RULE163
PURSUE-HYPOTHESIS [
 MYCOBACTERIUM-TB-MENINGITIS ]
RULE171
TEST-HYPOTHESIS [
 MYCOBACTERIUM-TB-MENINGITIS ]
RULE603
APPLYRULES [ RULE366 RULE309 RULE309
 RULE002 RULE525 ]
RULE094
APPLYRULE! [ RULE309 ]
RULE095
FINDOUT [ STEROIDS ]
RULE153
FINDOUT [ IMMUNOSUPPRESSED ]
RULE153
FINDOUT [ SYSTEMIC-COMPROMISED ]
RULE153
FINDOUT [ COMPROMISED ]
RULE169
```

Figure **4-** 13: The Task Stack Window.

As usual, rules, findings, hypotheses, and tasks are selectable. Additionally, the user can select an invocation number in order to display more information on the history of that task call, including a dynamic task tree with the chosen task invocation as the root or any metarules that succeeded during the task call.

Together, the three dynamic task windows provide a powerful aid for inspecting the current

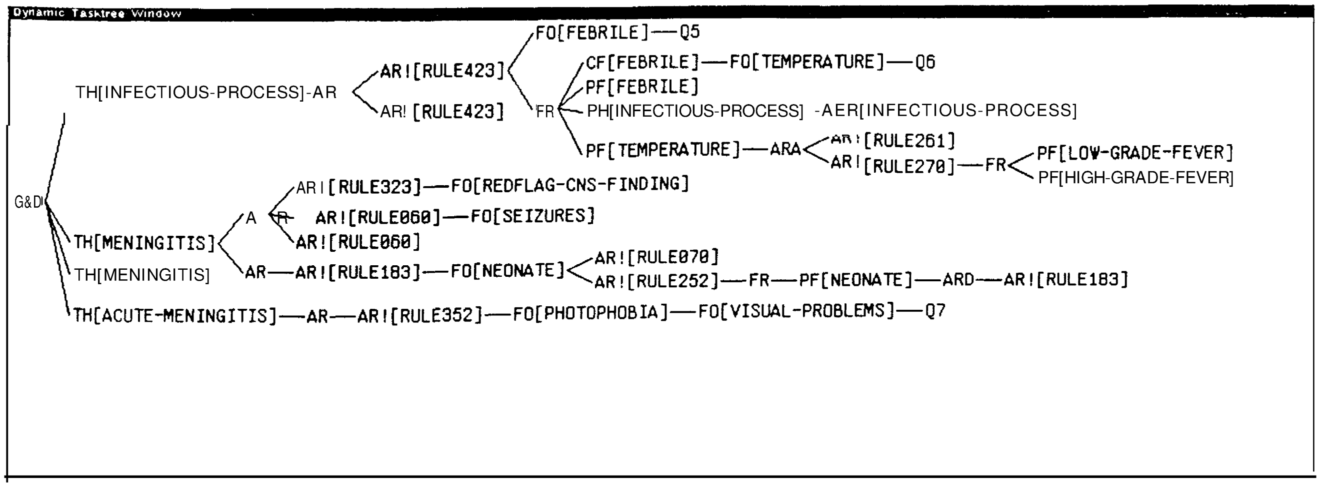

Dynamic Tasktree Window
G&D
TH[INFECTIOUS-PROCESS]-AR
AR![RULE423]
AR! [RULE423]
FO[FEBRILE]—Q5
FR
CF[FEBRILE]—FO[TEMPERATURE]—Q6
PF[FEBRILE]
PH[INFECTIOUS-PROCESS] -AER[INFECTIOUS-PROCESS]
PF[TEMPERATURE]—ARA
AR![RULE261]
AR![RULE270]—FR
PF[LOW-GRADE-FEVER]
PF[HIGH-GRADE-FEVER]
TH[MENINGITIS]
A
AR![RULE323]—FO[REDFLAG-CNS-FINDING]
AR![RULE060]—FO[SEIZURES]
AR![RULE060]
TH[MENINGITIS]
AR—AR![RULE183]—FO[NEONATE]
AR![RULE070]
AR![RULE252]—FR—PF[NEONATE]—ARD—AR![RULE183]
TH[ACUTE-MENINGITIS]—AR—AR![RULE352]—FO[PHOTOPHOBIA]—FO[VISUAL-PROBLEMS]—Q7

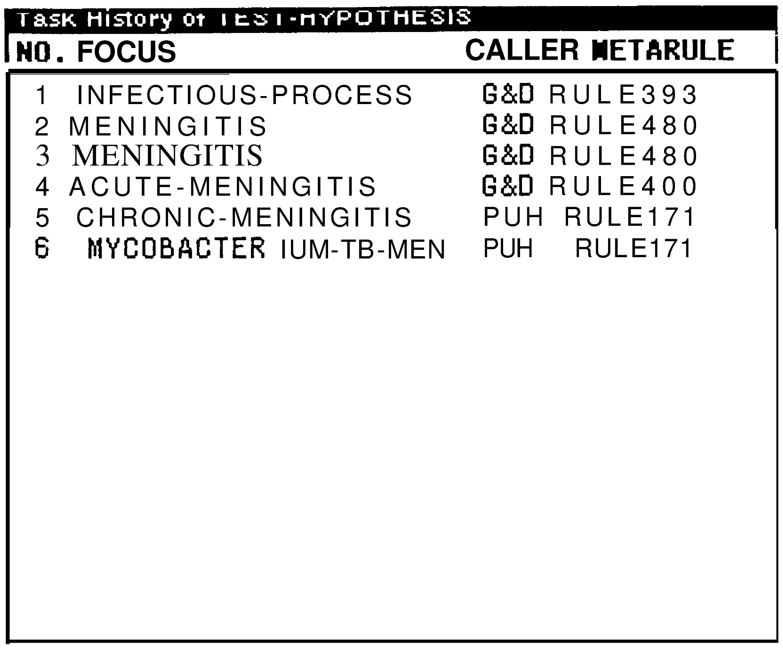
Figure 4- 14: The Dynamic Task Tree Window.

*The node labels of tasks in this tree are abbreviated, but the user can see the full name expanded when a node is selected. TH is short for Test-Hypothesis in this tree. The figure above illustrates how the user can see multiple calls of a task and the resulting events in the Dynamic Task Tree Window.*

and past diagnostic strategy used during the consultation. It is clear that medical students would need some instruction in these concepts before the dynamic **task** windows would be meaningful to them. Part of the problem is that many of the task names are not commonly used in medicine. It is hoped that some of the problem can be alleviated by choosing names for the diagnostic tasks that are more familiar to students. Additionally, some tasks may be hidden from a student's view because they involve computational details of interest to a programmer only.

## 5. Prior and Related Work in Graphic Interfaces

GUIDON-WATCH was influenced by a diverse collection of work stretching back to Vannevar Bush's seminal article, in which a desk-sized, electronic information device called a "memex" was proposed (Bush, 1945). Doug Engelbart and his colleagues pioneered much of the early work on information handling systems and provided the basis for the user interfaces commonly found on today's workstations (Engelbart, 1963, English and Engelbart, 1967, Engelbart, 1970, Engelbart, 1982). Alan Kay led the Learning Research Group (LRG) at



Task History of TEST-HYPOTHESIS

| NO. | FOCUS | CALLER | METARULE |
|---|---|---|---|
| 1 | INFECTIOUS-PROCESS | G&D | RULE393 |
| 2 | MENINGITIS | G&D | RULE480 |
| 3 | MENINGITIS | G&D | RULE480 |
| 4 | ACUTE-MENINGITIS | G&D | RULE400 |
| 5 | CHRONIC-MENINGITIS | PUH | RULE171 |
| 6 | MYCOBACTERIUM-TB-MEN | PUH | RULE171 |

Figure 4-15: The Task History Window.

XEROX PARC that brought similar ideas to fruition on personal workstations. (Kay, 1977) Engelbart's group and the LRG shaped a view of the computer as a communications medium by which a user can store, retrieve, manipulate, and transfer information with ease. The LRG's vision of a dynabook (Goldberg, 1979, Borning, 1979, Weyer and Borning, 1984, Gould and Finzer, 1984) stills remains an exciting dream in the spirit of Bush's memex.

The 1970s also brought many advances in Artificial Intelligence including the development of knowledge-based systems such as MYCIN. Starting from a different point of view, Seymour Papert led a group at M.I.T. that explored the use of computer languages such as LOGO to teach subjects such as geometry and physics in a new way (Papert, 1980). Papert offers a provocative view of AI and computers in education; he influenced us to consider how we can provide students with conceptual and software tools to explore computational models (Papert, 1980). John Seely Brown further inspired us to understand the potential of these ideas; his discussion of *reifying the process* of problem solving (Brown, 1983) is particularly relevant to GUIDON-WATCH. To *reify* means to make real or concrete, or to materialize something that is abstract. GUIDON-WATCH makes the abstract diagnostic procedure used in NEOMYCIN more concrete and visible.

Partly because bit-mapped raster displays have only recently been integrated with AI programming environments, little has been written about graphic interfaces in AI programs. Some notable exceptions include Model (Model, 1979), AIPS (Zdybel, et al., 1981), the ONCOCIN project (Tsuji and Shortliffe, 1983, Tsuji and Shortliffe, 1985, Lane, et al., 1985), and the STEAMER project (Hollan, et al., 1984, Stevens, et al., 1983). Model demonstrated that graphic displays can facilitate the monitoring and debugging of complex programs (he used MYCIN for an early demonstration of his work). Tsuji and Shortliffe investigated this idea further by implementing several graphic tools for constructing, monitoring, and debugging ONCOCIN's knowledge base and inference procedures. (ONCOCIN is a system that helps physicians administer experimental cancer therapy.) The ONCOCIN group's strong commitment to the use of interactive graphics has resulted in several graphic interfaces, including the ONCOCIN *Interviewer* (Lane, et al., 1985), a program that helps physicians enter patient data.

STEAMER, an instructional program about power plant operation, uses interactive color graphics in a knowledge-based simulation. It is interesting to compare the use of interactive graphics in STEAMER and GUIDON-WATCH. STEAMER emphasizes the construction of a *visible, interactive, and inspectable simulation.* It displays the complex *physical processes* of a steam propulsion plant. NEOMYCIN, on the other hand, is a *computational model of diagnostic reasoning.* GUIDON-WATCH provides a user with a visible, interactive, and inspectable model of NEOMYCIN's *reasoning processes.*

Several knowledge-base browsers similar to GUIDON-WATCH were developed more or less concurrently. For example, ART, KEE, SRL+, LOOPS, S.l, and STROBE provide interactive graphic displays that allow programmers to browse class hierarchies and other general data structures (Stefik, et al. , 1983, Kunz, et al., 84, Williams, 1984, Richer, 1985). Sophisticated graphic editors may be provided; for example, STROBE has an excellent knowledge base editor (Schoen and Smith, 1983). However, the browsers provided in these systems are very general and are too complex for end users, requiring an understanding of the underlying knowledge representation framework. On the other hand, KEE and LOOPS do provide support for creating end-user iconic displays. These can be very useful for displaying the state of a complex device.

GUIDON-WATCH differs from other browsing programs because it is tuned to display specific kinds of knowledge structures (e.g., those found in a HERACLES system). For example, GUIDON-WATCH can display disease taxonomies, causal networks, evidence for a hypothesis, and positive findings in a way that is *appropriate for end users* such as medical students. Graphic techniques described in this paper illustrate an abstract diagnostic procedure during its actual use. To paraphrase, if a knowledge base is written for HERACLES, then an effective user interface is provided automatically.

## 6. Future Work

A continuing decrease in the price of hardware will provide more opportunities to use higher resolution screens, interactive pictures, color, animation, and interactive video. Certainly, we have only touched the surface in using graphics for viewing a knowledge-based system. Interactive and animated pictures can illustrate facts and processes. However, implementing interactive graphic displays is time-consuming. There is a need for high-level user interface kits that provide most of the common features that developers now have to implement over and over again. It is probable that an object-oriented programming system will be adopted as an extension to the Common Lisp standard (Bobrow, et al. , 1985, Steele, 1984). This could 'provide the basis for a generic interface shell for lisp environments. Two examples of interface packages that successfully use the object-oriented approach include MacApp (Tesler, .1985) and EzWin (Lieberman, 1985); the latter is written in *flavors,* the object-oriented language within Zetalisp (Weinreb and Moon, 1981).

The discussion so far has focused on what interactive graphics can provide for AI systems. However, AI technology can contribute directly to more intelligent graphic interfaces. Current -research topics include user modeling, intelligent presentation, and declarative languages for describing graphical interaction. Mackinlay (Mackinlay, 1983) is investigating some of these issues. For GUIDON-WATCH, we decided how to present information and hand-coded it. Instead, Mackinlay's program *reasons on its own* about how to present information. For example, it can decide to present data as a bar chart, a pie chart, a plot chart, a table, or a graph. It can also design several alternative sophisticated presentations from simpler ones and then use heuristics to choose one to display to the user.

Another important aspect of Mackinlay's work is that it uses a knowledge-based approach. Therefore, its reasoning is represented in an explicit, declarative language and not in opaque code. The use of a declarative representation results in programs that are easier to modify and understand. We found that the parts of our display code that are trying to be *smart,* such as the management of windows, are poorly represented in LISP. The code fails to make the underlying reasoning explicit, and it is difficult to modify. Another advantage of using a declarative representation is that it can be used in multiple ways. Mackinlay's current work addresses only the intelligent presentation problem, but eventually programs may be able to explain why a particular presentation was chosen. The graphic designer using an intelligent computer aid would want a justification for some design decisions. In intelligent tutoring systems, it would be useful if a program could automatically generate questions regarding a presentation on the screen.

The problems involved in developing intelligent interfaces are certainly very difficult, but if they are going to be solved, it seems likely that user interface behavior must be represented separately in a declarative language or a data base. Because the user interface is becoming an increasingly complex and important component of a software system, there are compelling reasons to make a clean separation between the user interface and the rest of a software system (Zdybel, et al., 1981, Smith, et al., 1984, Ciccarelli, 1984). There are several reasons to support such highly modular systems:

- they are easier to maintain and debug
- they can be customized more easily
- domain independence is possible
- an intelligent reasoning component can be interfaced with less difficulty

In general, programs can be more attuned to individual users. Some users may prefer different configurations of the screen. The size of the fonts chosen in a window may be too small for some users. Optimizing screen space must not interfere with other concerns such as readability of the screen. Future versions of GUIDON-WATCH should allow users to customize the display to their liking while still providing automatic window management facilities. User models can play a role in smart interfaces that infer a user's preferences. However, a program must have an explicit model of the user in order to reason about the user's preferences. We believe that a knowledge-based approach (i.e., using declarative representations) is necessary if a intelligent interface must combine general knowledge about presentation with specific knowledge about a user. This is an area for long-term interdisciplinary research in several areas of computer science, psychology, linguistics, communications, education, and graphic design.

## 7. Conclusions

GUIDON-WATCH allows a user to view a knowledge-based consultation system efficiently. The program demonstrates how multiple windows, menus, and a mouse can be used to achieve this goal. It also demonstrates that stating a diagnostic procedure in a well-structured rule-based language facilitates developing a graphic interface for viewing and inspecting diagnostic problem-solving behavior. The most important principles learned from this effort are as follows:

- **Providing multiple views of the same knowledge or behavior can help a user understand a complex system.** Tables, trees, pictures, animation, and other graphic formats can offer these different views. The current prototype of GUIDON-WATCH has made extensive use of trees and tables to display information in multiple, meaningful ways. Hierarchical relationships are naturally represented as trees, and lists of records with several fields are displayed as tables effectively. There are several- important events in NEOMYCIN such as changes in the differential, conclusions about findings and hypotheses, and the task calls. Several windows with different formats can provide different views of these events. However, different classes of users may vary with regard to what constitutes an effective user interface.

- **The use of bold fonts, boxing and graying items, and other graphic techniques can maximize information content and highlight facts and events in a way that is quickly understandable.** The use of these simple techniques in the Taxonomy and Evidence windows illustrates their effectiveness (Figures 4-7 and 4-8).

- **In well-constrained situations it is possible to manage the display and placement of windows automatically.** Screen space is a precious resource, and each window must be 'designed, sized and placed to use space efficiently. However, this is a job that can be cumbersome for a user. Additionally, we want to avoid having a user concentrate on the motor activity of using the mouse to move and place windows on the screen. We believe that there is a fundamental difference between the information retrieval task that GUIDON-WATCH is designed for and more creative and open-ended tasks such as programming or writing. For the latter category, the availability of overlapping windows that are usually shaped and positioned under the user's control may be more desirable.

By displaying information in multiple ways and allowing a user to interactively browse the dynamic state of a consultation, we have taken a first step towards reifying the process of reasoning during a NEOMYCIN consultation. Subsequent instructional programs now under development will ask students to explain, debug, and augment their own and program-generated problem-solving behavior. They will use graphic displays like GUIDON-WATCH to compare and contrast alternative solutions to problems.

## Acknowledgements

Diane Warner Hasling, Julie Thompson Richer, Ted Crovello, and the CG&A reviewers contributed valuable comments to earlier drafts of this paper. Dr. Curt Kapsner and John Macias are helping us adapt NEOMCYIN and GUIDON-WATCH for use with medical students. Their feedback is helping us determine how GUIDON-WATCH and other programs can best benefit medical students.

This research has been supported in part by ONR and ARI Contract N00014-79C-0302 and more recently by the Josiah Macy, Jr. Foundation (Grant No. B852005). Computational resources have been provided by the SUMEX-AIM facility (NIH grant RR00785).

## References

Bobrow, D.G., Kahn, K., Kiczales, G., Masinter, L., Stefik, M., and Zdybel, F. *COMMONLOOPS -- merging COMMON LISP and Object-Oriented Programming.* Technical Report ISL-85-8, XEROX Palo Alto Research Center, August 1985.

Borning, A. *Thinglab -- a constraint-oriented simulation laboratory.* Technical Report SSL-79-3, Xerox Palo Alto Research Center, July 1979.

Brown, J.S. *Process versus product--a perspective on tools for communal and informal electronic learning,* in *Education in the Electronic Age,* July, 1983. Proceedings of a conference sponsored by the Educational Broadcasting Corporation, WNET/Thirteen Learning Lab, NY, pp. 41-58 .

Buchanan, B.G. and Shortliffe, E.H. *Rule-Based Expert Systems.* Reading, MA: Addison-Wesley 1984.

Bush. As we may think. *Atlantic Monthly,* July 1945, 176, 101-108.

Card, S.K., Moran, S.K., and Newell, A. *The Psychology of Human-Computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates 1983.

Ciccarelli, E. *Presentation based user interfaces.* Technical Report AT-TR-794, Artificial Intelligence Laboratory, Massachusetts Institute Technology, August 1984.

Clancey, W.J. Overview of Guidon. *Journal of Computer-Based Instruction,* Summer 1983, *10(1 & 2),* 8-15. (Also in *The Handbook of Artificial Intelligence,* Volume 2, eds. Barr and Feigenbaum, Kaufmann, Los Altos, CA, 1982, pp. 267-278.).

Clancey, W. J. and Letsinger, R. NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching. In Clancey, W. J. and Shortliffe, E. H. (editors), *Readings in Medical Artificial Intelligence: The First Decade,* pages 361-381. Addison-Wesley, Reading, MA, 1984.

Engelbart, D.C. A conceptual framework for the augmentation of man's intellect. In Howerton and Weeks (editors), *Vistas in Information Handling,* pages 1-29. Spartan Books, Washington, D.C., 1963.

Engelbart, D. *Advanced intellect-augmentation techniques.* SRI project 7079, final report, Stanford Research Institute, July 1970.

Engelbart, D.C. *Toward high-performance knowledge workers,* in *Proceedings of the AFIPS Office Automation Conference,* pages 279-290, April, 1982.

English, W.K. and Engelbart, D.C. Display-selection techniques for text manipulation. *IEEE Transactions on Human Factors in Electronics,* March 1967, *HFE-8(1),* 5-15.

Foley, J.D. & Van Dam, A. *Fundamentals of Interactive Computer Graphics.* Reading, MA: Addison-Wesley 1982.

Foley, J.D., Wallace V.L., and Chan, P. The human factors of computer graphics interaction techniques. *IEEE Computer Graphics and Applications,* November 1984, *4(11),* 13-49.

Goldberg. Educational uses of a dynabook. *Computers & Education,* 1979, 3, 247-266.

Gould, L. and Finzer, W. *Programming by rehearsal.* Technical Report SCL-84-1, Xerox Palo

Alto Research Center, May 1984.

Hasling, D. W., Clancey. W. J., and Rennels, G. Strategic explanations for a diagnostic consultation system. *International Journal of Man-Machine Studies,* 1984, 20, 3-19.

Hollan, J. D., Hutchins, E. L., and Weitzman, L. STEAMER: An interactive inspectable simulation-based training system. *The AI Magazine,* 1984, *5(2),* 15-27.

Ingalls, D.H.H. Design principles behind smalltalk. *Byte,* August 1981, *6(8),* 286-298.

Kay, A. Microelectronics and the personal computer. *Scientific American,* September 1977, *237(3), 230-244.*

Kunz, J.C., Kehler, T.P. and Williams, M.D. Applications development using a hybrid AI development system. *AI Magazine,* Fall 84, *5(3),* 41-54.

Lane, C., Differding, J., and Shortliffe, E. *Graphical access to medical expert systems: II. Design of an interface for physicians.* Technical Report KSL-85-15, Knowledge Systems Laboratory, Stanford University, July 1985.

Lieberman, H. There's more to menu systems than meets the screen. *Computer Graphics,* July 1985, *19(3),* 181-189.

Mackinlay, J. *Intelligent presentation: the generation problem for user interfaces.* Technical Report HPP-83-34, Computer Science Department, Stanford University, March 1983.

Model, M. *Monitoring system behavior in a complex computation environment.* Technical Report STAN-CS-79-701, Computer Science Department, Stanford University, January 1979.

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

Richer, M.H. *Evaluating the existing tools for developing knowledge-based systems.* Technical Report KSL-85-19, Knowledge Systems Laboratory, Stanford University, May 1985.

Sannella, M., ed. *Interlisp Reference Manual.* Palo Alto, CA: Xerox Corporation 1983.

Schoen, E. and Smith R.G. *Impulse, a display-oriented editor for Strobe,* in *Proceedings of the National Conference on AI,* pages 356-358, AAAI, August, 1983.

Shortliffe, E. H. *Computer-based medical consultations: MYCIN.* New York: Elsevier 1976.

Smith, R.G., Lafue, G.M.E., Schoen, E. and Vestal, S.C. Declarative task description as a user-interface structuring mechanism. *Computer,* September 1984, *17(9),* 29-38.

Steele, G.L. *Common LISP -- the language.* Burlington, MA: Digital Press 1984.

Stefik, M., Bobrow, D.G., Mittal, S. and Conway, L. Knowledge programming in loops: report on an experimental course. *The AI Magazine,* Fall 1983, *4(3),* 3-13.

Stevens, A., Roberts, B., and Stead, L. The use of a sophisticated graphics interface in computer-assisted instruction. *IEEE Computer Graphics and Applications,* March/April 1983, *3(2),* 25-31.

Tesler, L. The smalltalk environment. *Byte,* August 1981, *6(8),* 90-147.

Tesler, L. *MacApp, Release 0. I.* Cupertino, CA: Apple Computer, Inc. 1985.

Thompson, T.F. and Clancey, W.J. *The CASTER system: an experiment in knowledge acquisition within a generic expert system shell.* Technical Report KSL-85-32, Knowledge Systems Laboratory, Stanford University, August 1985.

Tsuji, S. and Shortliffe, E. *Graphical access to the knowledge base of a medical consultation system,* in *Proceedings of AAMSI (American Association for Medical Systems and Informatics) Congress 83,* pages 551-555, May, 1983.

Tsuji, S. and Shortliffe, E.H. *Graphical access to medical expert systems: I. Design of a knowledge engineer's interface.* Technical Report KSL-85-11, Knowledge Systems Laboratory, Stanford University, July 1985.

Van Melle, V. *System Aids in Constructing Consultation Programs.* Ann Arbor, Michigan: UMI Research Press 1981.

Weinreb, D. and Moon, D. *Lisp Machine Manual.* Cambridge, MA: Symbolics, Inc. 1981.

Weyer, S. and Borning, A. *A prototype electronic encyclopedia.* Technical Report 84-08-01, Computer Science Department, University of Washington, August 1984.

Williams, C. Software tool packages the expertise needed to build expert systems. *Electronic Design,* August 1984, 153-167.

Zdybel, F., Greenfield, N., Yonke, M. and Gibbons, J. *An Information Presentation System,* in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* pages 978-984, August, 1981.