Report 85-20
Stanford -- KSL

Scientific DataLink

GUIDON WATCH: A Graphic Interface for Browsing and Viewing a Knowledge-Based System. Mark H. Richer, William J. Clancey, May 1985

card 1 of 1

Knowledge Systems Laboratory

Report No. KSL 85-20

May 1985

GUIDON WATCH:
A Graphic Interface for Browsing and Viewing a Knowledge-Based System

by

Mark H. Richer

William J. Clancey

Knowledge Systems Laboratory
(incorporating the Heuristic Programming Project)
Stanford University
701 Welch Road (Building C)
Palo Alto, CA 94304

Submitted for review: IEEE Computer Graphics & Applications

Introduction

An increasing number of Artificial Intelligence (AI) programs are implemented on high-performance workstations with a bitmap display, a mouse input device, and a keyboard. The programming environment (usually a dialect of LISP) generally provides support for multiple, overlapping windows, and various kinds of menus including pop up menus. The user can move, reshape, close, and scroll the windows. Additionally, a programmer can designate arbitrary regions of a window to be selectable with the mouse. This means that a user can invoke an action by pressing and releasing a mouse button while the mouse cursor is in the designated region. (We say a user *buttons* or *selects* an item with the mouse.)

The capability to display information in several windows and the availabiltiy of a mouse device provides many opportunities to use interactive graphics for developing improved user-interfaces. Graphic interfaces can be used to help system designers and programmers construct, monitor and debug complex programs. They can also be used to allow end-users to control and observe the behavior of an application program.

GUIDON-WATCH. The program described in this paper is called GUIDON-WATCH; it is the first component in a new family of instructional programs. (The name is derived from GUIDON[1], a tutoring program based on the MYCIN[2] consultation program.) The goal of these programs is to teach medical students how to diagnose diseases. GUIDON-WATCH provides a graphic interface to the NEOMYCIN[3] knowledge base and allows a user to view and inspect system behavior during a consultation.

NEOMYCIN. The foundation of the research described in this paper is NEOMYCIN, a program that diagnoses several diseases including meningitis. NEOMYCIN is a reconfiguration of MYCIN that is designed to serve more appropriately as a

foundation for instructional programs. NEOMYCIN differs from MYCIN in that a domain-independent diagnostic procedure is represented separately from medical facts in an explicit language that facilitates explanation and student modeling. Additionally, the original knowledge base has been significantly broadened. The goal of the research is to investigate strategies for teaching diagnosis to students.

The representation of knowledge in MYCIN is inadequate for teaching programs because it is not possible to make clear to students the meaning of individual rules nor the overall diagnostic strategy. MYCIN rules combine domain facts with a diagnostic procedure in an opaque manner. Importantly, interactive graphics can make knowledge and reasoning visible only to the extent that the knowledge is represented *explicity* 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.

HERACLES. NEOMYCIN has been generalized to HERACLES in much the same way that MYCIN led to EMYCIN[4]. HERACLES is a software tool applicable to diagnostic tasks in many domains. For example, HERACLES is being used to develop a knowledge base for cast iron fault diagnosis. The HERACLES program provides a knowledge engineer with a diagnostic procedure represented in a declarative language, a domain rule interpreter, a set of domain relations (e.g., Causes), various software tools for developing knowledge-based systems (many derived from EMYCIN), an explanation facility, and GUIDON-WATCH, a domain-independent graphic interface to HERACLES. To construct a specific consultation program, the system designer adds a knowledge base of facts and rules. (figure 1) All the examples in this paper use the NEOMYCIN knowledge base, but GUIDON-WATCH will work analogously for any HERACLES knowledge base.

General Considerations. Graphic interfaces to knowledge-based systems can serve several different classes of users including system designers, implementers, domain experts in the role of collaborator or end-user, students, and other end-users. Because the goal of the NEOMYCIN project is to develop instructional programs, our bias has been to create a display system that will help students understand NEOMYCIN's diagnostic reasoning. The program described in this paper is a prototype that makes NEOMYCIN's knowledge and reasoning processes visible to a student. However, to effectively use the program described here requires some knowledge of computing and AI concepts; therefore, the current prototype is only a first step. We believe it is possible, even advantageous, to teach medical students some basic AI, computing, and cognitive science concepts. GUIDON-WATCH will eventually include an introduction to the basic components and terminology of NEOMYCIN.

Prior work. GUIDON-WATCH was influenced by a variety of work stretching back to Vanevar Bush's seminal article, in which a desk-sized, electronic information device called a "memex" was proposed.[5] The research of Englebart[6], Kay[7], Goldberg[8], and others have further shaped a view of the computer as a communications medium by which a user can store, retrieve, manipulate and transfer information with ease. Papert's provocative view of the role of computer technology and AI in education influenced us to consider how we can provide students with conceptual and software tools to explore computational models[9]. Papert suggests that students can and should be taught computational models of cognition. For example, the HERACLES diagnostic operators can be used as a notation for describing diagnostic strategies. John Seely Brown further inspired us to understand the potential of these ideas; his discussion of *reifying the process* of problem solving[10] 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 concrete and visible.

Partly because bit-mapped raster displays have only recently been integrated with AI programming environments, little has been published about the use of interactive graphics in Artifical Intelligence programs. Some notable exceptions include the work of Model[11], the ONCOCIN project (Tsuji and Shortliffe[12], and Lane, et. al[13]), and the STEAMER group (Hollan, et. al.[14] and Stevens, et. al.[15]). Model demonstrated that graphic displays can facilitate the creation of tools for monitoring and debugging complex programs. 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. Shortliffe's strong commitment to the use of interactive graphics has resulted in several graphics-based interfaces, including the *ONCOCIN Interviewer*, a program that helps physicians enter patient data.

STEAMER, an instructional program, 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 research emphasizes the construction of a *visible, interactive, and inspectable simulation.* STEAMER 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.*

Graphics Hardware. GUIDON-WATCH is implemented on XEROX 1100 series workstations running INTERLISP-D. The display screen is 1024 pixels wide by 808 pixels high, which provides approximately 75 dots per inch resolution. A three-

button mouse is used with the right button reserved for window operations such as reshaping, moving, and closing windows.

Design Criteria for GUIDON-WATCH

The designer of a graphic interface is creating a means of two-way communication between a person and a program. One typical model is that a user inputs requests with a keyboard, mouse, or some other input device, and the program responds with output on the display screen or some other device. Therefore, a user interface can consist of two languages; the language by which the user makes requests and the language that the program uses to make responses. In this sense, interaction with a mouse and menus constitutes a language, and the display of information in a window constitutes another language. In the first case, a user is concerned with the process of *retrieval* and in the second case, with the process of *interpretation*. In both cases we can evaluate the languages with regard to *expressiveness*, *understandability*, and *efficiency*.

Given our instructional objectives, we decided that the resulting program must meet the following criteria:

With Regard To Retrieval

- **Expressiveness.** The expressiveness of a query language can be judged by *what* a user can request with the language. A language is adequately expressive if a user can make desired requests. In this context, we will say that a language is *complete* if a person can use the language to request any information that the system is capable of providing. Our goal is to allow a user to make *all* requests through a simple, uniform interface. In GUIDON-WATCH, the interface should allow a user to view any part of the knowledge base or the program's reasoning processes.

- **Understandability.** An understandable query language is

one in which the user can understand *how* to make requests, and the computer interprets these requests correctly. We equate a *transparent* user-interface with an understandable query language. A transparent interface permits a user to expend a miminum amount of energy on *how to retrieve* information. This requires making the interface simple and consistent. An interface can be fully expressive and yet not fully transparent when a request can only be made by using the interface in a way that is awkward.

- **Efficiency.** The efficiency of a querry language refers to the time it takes to make a request once it has been formulated in the language. This is concerned with the *mechanics* of making requests. Typically a user inputs information into a computer using a keyboard, a mouse, a light pen, a graphics tablet, voice, etc. Users of GUIDON-WATCH should be able to quickly browse the knowledge base or to select some behavior of the program to observe. We believe that the mouse is an excellent input device for a program such as GUIDON-WATCH. However, it is essential that the program respond quickly when a user presses a mouse button. For example, menus must pop up quickly.

  Efficiency can also be evaluated with regard to how many steps are required to make a request. A flexible interface allows a user to make a request at any time with a minimal number of steps. In GUIDON-WATCH, we want a user to be able to request information at any time without having to switch into another mode. Additionally, we want to minimize how often the user needs to move the mouse or press mouse buttons.

With Regard To Interpretation

- **Expressiveness.** It is essential that the language chosen for responding to a user's requests can appropriately express the *content* of the information that the user desires. GUIDON-WATCH must provide a user with the information needed to understand NEOMCYIN's knowledge base and reasoning behavior. Furthermore, the program should provide *multiple views* of the same knowledge and behavior. This provides students and other users with more opportunities to understand the program's behavior. To meet this goal, the availability of multiple windows can be exploited to provide a variety of information on the screen at one time.

- **Understandability.** The user should be able to understand a program's output as easily as possible. The program must organize information at several levels of detail so that a user can control the amount and detail of information displayed. Furthermore, information should be displayed in *recognizable* formats such as tables, lists, graphs, etc. Techniques such as flashing, boxing and graying regions and using bold fonts can be used to highlight important facts and events, but the user must be aware of their meaning. In some cases, instruction on using the program or a help facility may be necessary, and unavoidable. However, a user should not be required to understand the underlying details of a system in order to understand the output, any more than to generate a request. In this sense, the meaning of the program's output should be *transparent*.

- **Efficiency.** Presentations should not only be

understandable, but should also be *concise* and highlight important information so as to make interpretation faster. Text should be readable, and the layout in general should conform to standards of graphic design. Of course, the program *response time* should be fast so that text and graphics are displayed quickly on the screen.

Description of the GUIDON-WATCH display

The various menus and windows in GUIDON-WATCH are described in detail in this section. We focus on the following questions: (1) Which information in a HERACLES knowledge base is most important to display? and (2) How can this information be displayed?

Pull down menus. (figure 2) In the top left corner of the screen are several pull down menus. When the user selects a menu item, a second menu is displayed directly underneath the first. (The idea for using pull down menus came from the Apple LISA/Macintosh pull down menu bar.) The use of pull down menus saves screen space and hides unneccessary details from a user. A user can use the pull down menu to display windows, start a consultation, get help, and execute system utilities.

General Structure of the Display. (figure 3) Many of the windows in GUIDON-WATCH can be displayed by selecting an item from the *KB Windows* pull down menu. A user uses the mouse to select items in these windows. As a result, a pop up menu is displayed that allows the user to print more information in other windows. Items in these windows can also be selected to display a pop up menu. The contents of a window can be a list, a graph, a table, a stack, or a property list. Windows can display static knowledge structures, dynamic information, or a combination of both.

Static knowledge base windows. The primary objects in a

HERACLES knowledge base include findings (e.g., headache), hypotheses (e.g., meningitis), rules, tasks (e.g., test-hypothesis), and relations (e.g., kind-of). Static knowledge consists of facts about each object including the relationships between objects as defined by binary relations (e.g., X causes Y), disorder hierarchies, domain rules (e.g., if the patient has double vision, then there is suggestive evidence for intracranial pressure), and the diagnostic procedure. 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.").

Windows that display static knowledge include the Findings, Hypotheses (figure 2), Tasks, and Relations windows, which simply display an alphabetical list of the objects. Another set of windows uses graphs to show the relationships between groups of objects. The Causal Association window (figure 4) is a lattice with causal and subtype links between findings and hypotheses; the Metastrategy window (figure 5) shows the calling structure of the diagnostic tasks; and the Taxonomy window (figure 6) represents a subtype hierarchy of disorders. Graphs, in contrast with traditional, linear teletype-style output, provide an excellent way to display a hierarchy as a spatial ordering.

Selectable items. (figure 5) Knowledge base objects are selectable when they appear in knowledge base windows. When a user buttons an object, a pop up menu is displayed with choices relevant to the selected object. Therefore, buttoning an item allows the user to get additional information about the object selected. During a consultation, additional items appear on the pop up menus that allow a user to display dynamic information structures.

Hypotheses. Buttoning a hypothesis pops up a menu that may include the items *evidence*, *causal relations*, and *subsumption relations* Subsumption relations include subtype relations and attribute relations. For example, the existence of a headache in a

patient subsumes the duration or severity of the headache. By selecting a hypothesis in a window a user can quickly obtain further information by displaying the Causal Relations, Subsumption, and Evidence windows (all in figure 2).

Findings. A pop up menu for findings may include the items *causal relations*, *subsumption relations*, and *suggests*. If the user selects *suggests* then the hypotheses that the finding suggests are printed in the Prompt window.

Tasks. Tasks are procedures that are stated in a declarative language. When a task is invoked, one or more of its metarules are applied. Metarules in HERACLES are similar to conditionals in a procedure, but they are expressed as abstract rules. For example, a metarule translated into English could be "when gathering data, if there is a more general finding than the one you are interested in, then consider the more general one first." In a medical context, this could imply that the program should ask about surgery before cardiac surgery.

Each task is associated with a list of metarules and other properties that define the task's control structure. Tasks can be *simple* or *iterative*, indicating whether or not metarules are applied repeatedly. When a task is selected in a window, a menu pops up that allows the user to display either the properties of the task in the Task Property window, or the metarules that a task applies in the Metarules window (both windows in figure 5). During a consultation the user can also choose to see dynamic information about task calls. This is described in the section **Dynamic Task Windows.**

Rules. (figure 2) Domain rules and metarules in HERACLES consist of a premise, action, and several other properties. When a rule is buttoned in a window, its premise, action, and some other properties are printed in the Rule window. Unlike other objects in HERACLES, when a rule is selected, a pop up menu is not

displayed. Current users of the system prefer to see the rule's definition immediately because a symbol such as *RULE424* is meaningless to a person. This is a case where convenience seems more important than maintaining consistency. To increase flexibility, a user can also select *Rules* from the *KB Windows* menu. In that case, the user is prompted for a rule number.

The Rule window has a pop up menu associated with it that allows a user to see additional information about the rule in an auxillary window. This information includes the author's name, last edit date, justification, and translations. Menu items for translations include *English*, *LISP code*, and *Terse* English.

Window menus. In the windows described above, the user selects an object to see more details about it. In addition, there could be information about the entire structure (e.g., causal network) that is of interest to the user. For example, a user may want to know which of the disorders in the Causal Association network have evidence, causal or subsumption relations. If the user releases the mouse button in a window with a graph, and the mouse cursor is *not* in a nodelabel region, then a pop up menu is displayed. (The nodelabel is the text string printed at a node position.) In windows with hypotheses, the menu items include *evidence*, *causes*, etc. If the user chooses *causes* then all the disorders that have causes (that are represented in the knowledge base) are inverted. (figure 4) If the user chooses *evidence* from the pop up menu in the Hypotheses window, then it is possible to quickly see all the disorders that can or can not be concluded directly from rules. Many of the task windows have a pop up menu that allows a user to see which tasks are *simple*, *iterative*, etc.

Reifying the process. Displaying a hierarchy of disorders as a tree allows a user to quickly see the structure of the solution space and to browse the knowledge base to see the rules that can conclude about a hypothesis. The Metastrategy window displays the static calling structure of the tasks in the diagnostic

procedure. However, these windows do not give a user any leverage in *viewing the dynamic search process*. NEOMYCIN's diagnostic procedure is a method for searching hierarchically organized solution spaces efficiently. To reify this reasoning, several interactive graphic displays are available to a user.

The *differential* (a set of competing hypotheses) represents a *cut* through the solution space, but a mere listing of hypotheses does not reveal the relationship between the hypotheses and the entire disorder hierarchy. A way of dynamically illustrating both the search process and the cut that the differential makes in the search space is illustrated in Figure 6. (Assume the Taxonomy or the Causal Association window is open). Whenever a hypothesis is added to the differential, its corresponding nodelabel in the Taxonomy and Causal Association windows is boxed. Whenever the hypothesis is removed from the differential, the box is redrawn with lighter lines, so that hypotheses that have been considered are shown, but the ones on the differential are more highlighted. This graphic notation makes the search path through the solution space more visible. Furthermore, the nodelabel is *flashed* when a hypothesis is added to the differential. As a result of these techniques, a fundamental part of the diagnostic strategy is dynamically illustrated. 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. In HERACLES, each hypothesis has both a certainty factor (CF) and a cumulative certainty factor (CUMCF). The CF represents the combined certainty of all rules that have concluded directly about the hypothesis. The CUMCF represents a combination of the CF (which may be zero) of a given hypothesis with CFs from other hypotheses, above and below it, 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. When the CF or CUMCF is updated for a hypothesis, new values are printed below the nodelabel corresponding to the hypothesis. The CF is printed on the left, the CUMCF on the right. In the case in which the window is not open, the printing, boxing and flashing of nodes is not done immediately. However, whenever the Taxonomy or Causal Association window is opened during a consultation, the window is updated so that all the hypotheses are appropriately boxed, and certainty factors are printed (not necessarily in the order the events actually occured). Therefore, the user is free to open these windows at any time.

We believe that what is most interesting pedagogically about a HERACLES system is not simply the static domain facts and relations that are represented in the knowledge base, but the *instantiation* of the abstract diagnsotic procedure during a consultation. The diagnostic procedure determines when domain knowledge is used. On the other hand, the domain knowledge is used to instantiate variables in the metarules. For example, the decision to test a hypothesis is represented in the diagnostic procedure, but the actual domain rules that are applied, if any, are represented in the domain knowledge base. Therefore, the abstract diagnostic procedure and the domain knowledge have meaning in relation to each other only through their application in a consultation. The boxing, flashing, and printing techniques described above makes this relationship visible in GUIDON-WATCH by displaying dynamic information on top of static knowledge structures, such as the disorder taxonomy and causal network.

Dynamic knowledge structures. Dynamic knowledge includes the differential and the certainty factors described above. It also refers to the values that have been determined for findings, and the current state and history of the task invocations. During a

consultation, additional windows that display only dynamic knowledge can be opened. The *KB Windows* pull down menu is augmented to include the Positive Findings, Differential, Hypotheses-With-Evidence, Task Stack, Task History Tree, and Task History windows.

The Consultation Typescript. (figure 7) The Consult window 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 consultation menu is provided that allows a user to proceed one or more questions further, receive textual explanations[16], resize the consultation window, etc.

To further reify the diagnostic process, it is necessary to make visible the *data gathering process*. One view of data gathering is represented in the consultation typescript. However, this is not fine-grained enough because inferences about findings and hypotheses are not shown. Tracing features in knowledge-based systems that print out inferences mixed with the typescript provide an overwhelming and disorganized array of information. The windows described below provide a view of data gathering, the pursuing of hypotheses, and the instantiated diagnostic strategy.

Evidence Window. (figure 7) This window can be displayed without running a consultation, but it takes on new meaning as dynamic information is gathered. Evidence for a hypothesis is displayed as a table in this window. The first column lists findings and hypotheses that are evidence for the given hypothesis. The second column lists the rules that use these findings or hypotheses to conclude about the given 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 information in a simple, 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. (figure 7) The Positive Findings window 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 (not printed if equal to "yes"), and CFs in the third (printed only if less than 1000). The findings in this window include all information offered by the user and findings that were concluded by rules. Findings are selectable, and when buttoned a pop up menu is displayed. Note that 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 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. (Figure 7) Hypotheses that are on the differential are boxed when they appear in certain windows. However, the differential is such an important structure that a separate window is provided to display these hypotheses. Another important class of hypotheses are 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 threshold).

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, the Taxonomy window, and the Causal Association window illustrate how GUIDON-WATCH provides *multiple views of the same knowledge structures.*

Dynamic Task Windows. The last set of windows described are designed to provide a user with a dynamic view of the diagnostic strategy as it is instantiated during a consultation. This is challenging 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 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 it 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, and also access global data structures. Tasks invoke other tasks in a chain, similar to procedure calls. In another sense, tasks can be viewed as operators that change the state of the consultation; in particular, the differential may be updated. Therefore, it is expected that students will need some instruction in these concepts before the dynamic task windows are really useful to them instructionally. On the other hand, as system builders we find the dynamic task windows to be excellent aids in monitoring and debugging the complex behavior of NEOMCYIN. These windows replace the fifty-page traces

previously used for debugging the system. The three windows described below each provide a different view of the dynamic diagnostic procedure. Furthermore, they illustrate the use of three different graphic techniques: a stack, a tree, and a table.

Task Stack Window. (Figure 8) This window displays the current stack of task calls, which is similar to a stack of procedure calls. The current design of this window shows the tasks 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. A focus consists of one or more findings, hypotheses, or rules. The metarule that the task successfully applied is printed below the task. Metarules are *attached* to the task they invoke by a vertical line. Different font features 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 metarules that invoked a task, the user can understand the reason for the current strategy. The display of the foci provide another view of their use during the consultation. For example, the user might be aware that the program has become interested in a particular disorder based on the fact that the disorder has been added to the differential. The user may have learned this fact because the hypothesis was boxed in the taxonomy window, it was added to the differential window, or the user inferred it from the consultation typescript. (Also, when a hypothesis is added to the differential the default is to print this information in the Prompt window.) However, the Task stack window provides a finer-grained view because the user can see exactly when and why a specific task (e.g., test hypothesis) was called with a certain hypothesis (e.g., meningitis).

Dynamic Task Tree. (Figure 8) In this window, the user can display a graph that shows all or part of the dynamic history of task calls. 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 Metastrategy window shows all possible paths in the task tree; this window shows only the paths that are part of an actual diagnosis.

Task History Window. (Figure 8) This window contains a table of all the invocations of any given task during the consultation. It provides an alternate view 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. As usual, rules, findings and hypotheses are selectable. Additionally, the user can select an invocation number in order to display more information on the history of that task call, including any metarules that succeeded during the task call. For the user's convenience, the window menu that is associated with the Task history window contains all the tasks that have been invoked during the consultation. In this way, a user can conveniently look at the history of several tasks without having to move the mouse to the *KB Windows* menu at the top of the screen.

User-Interaction in GUIDON-WATCH

An important part of the design of GUIDON-WATCH concerns user-interaction. Since the target users of systems such as NEOMYCIN and GUIDON-WATCH are not computer professionals, it is essential that the user-interface be simple to learn and use. It must provide the means for effectively using the program without knowledge of implementation details. User-interface issues that were addressed in the design and

implementation of GUIDON-WATCH are discussed in this section.

Interaction with the mouse. Almost all interaction in GUIDON-WATCH is achieved through the use of the mouse device. The mouse can simplify interaction with a computer, but attention still must be given to how the mouse is used. We believe that it is most important that the mouse be used in a simple and consistent manner.

There are tradeoffs concerning the use of a one, two, or three button mouse. A one-button mouse is very easy to use and the user never has to look at the mouse to figure out which button to press. Icons can be attached to a window so that a user can use the one mouse button to move, reshape, or close a window. Software can be written to distinguish between a single mouse press and two presses made in succession, allowing additional commands with the one button mouse. However , there are cases where a user can become confused about the effect of pressing a mouse button; it may result in one or more of the following actions: selecting an item, displaying a pop up menu, and bringing the window to the top of the display. Therefore, additional mouse buttons sometimes may provide more clarity as well as flexibility.

The XEROX 1100 series computers can be used with a two or three button mouse; pressing the left and right buttons at the same time is equivalent to pressing the middle button. The default action for the depressing the right button in a window is to pop up a menu that allows a user to move, reshape, and close the window. The left (and middle) button in GUIDON-WATCH is used to select items in a menu or knowledge base objects that appear in windows. When a user selects an object in a window, it is inverted and a pop up menu is displayed that applies to the selected object. Some windows can also have a pop up menu associated with the entire knowledge structure displayed (e.g., a

rule or taxonomy). These menus can be displayed by pressing the middle mouse button outside of a selectable region. The user can press the left button outside a selectable region to uncover a partially covered window. We found that using the left button for selecting items, popping up a menu, and uncovering a window is confusing; therefore, the middle button is used for popping up a window menu.

Dynamic Pop up Menus. It can confuse or irritate a user to be given a menu of items where some items are not currently relevant. Therefore, when a user selects a knowledge base object the pop up menu that is displayed is generated dynamically so that only applicable items appear on the menu. (An alternative is to "gray out" items that are not relevant; however, INTERLISP-D pop up menus do not support this feature.)

Help. After a user has depressed a mouse button over an INTERLISP-D menu item for a specified amount of time, a help string can be displayed in the INTERLISP-D Prompt window. This provides one type of on-line help. In addition, we added a *Help* pull down menu at the top of the screen that the user can use to get general help or information about a window. If the user selects *Help window* menu item, then the icon **HELP** 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 help window.

Management of windows. Windows in GUIDON-WATCH are pre-sized because the size and format of their contents is approximately known beforehand. Most windows that display detailed information are sized to about one-third of the screen width. They are placed side by side to use screen space in an optimal manner. When necessary, windows can be scrolled horizontally or vertically. With the current NEOMYCIN knowledge base, scrolling is seldom needed; for a very large

knowledge base, the contents of the windows could increase so much that scrolling would be inconvenient. For example, during a consultation, the number of entries in the Positive Findings window can outgrow the window. To accomodate these situations, many windows have a resize option on a pop up menu, which makes it easy to grow and shrink the window.

The GUIDON-WATCH screen is divided into three regions from top to bottom. The top part of the screen always includes the pull down menu and the Prompt window. The middle section of the screen is much larger in height and is used to display the Findings, Hypotheses, Tasks, Relations, Taxonomy, Dynamic Task Tree, and Metastrategy windows. The bottom section is smaller in height than the middle section and is generally used for displaying more detailed knowledge. The Causal Association, Metarules, Task, Task History, Positive Findings, Differential, and Hypotheses-With-Evidence windows are currently displayed in the bottom section. The other smaller windows, including the Rule, Evidence, Causal relations, and Subsumption windows may appear either in the bottom or middle section depending on the context. This knowledge has to be represented in the program, and at the current time it is represented in LISP, which can make it difficult to modify.

The arrangement of windows is based on considerations of size and which sets of windows a user will most likely want to display at once. For example, if the Taxonomy window is open and the user chooses to see the evidence for a hypothesis, then the Evidence window is opened on the bottom of the screen. Subsequently choosing to see a rule, or causal or subsumption relations, results in a window on the bottom being displayed. However, if the user chooses to see evidence for a hypothesis displayed in the Causal Association window (which is displayed in the bottom region), then the Evidence window is displayed in the middle section.

The automatic management of windows saves the user time, and is designed to fit the maximum amount of information on the screen at once. However, it increases the complexity of the implementation. In the present case, there are sufficient known constraints that make it possible to do this reasonably. Although a certain amount of flexibility and control is relinquished by the user, the benefits of automatic screen management seem to outweigh potential disadvantages.

Implementation Issues

GUIDON-WATCH was implemented on XEROX 1100 series workstations using INTERLISP-D. The INTERLISP-D programming environment provides a programmer with many primitives that faciliate the creation of multiple window displays. Therefore, we were able to concentrate more on developing our ideas, rather than on implementing low-level graphics tools. However, the event-driven nature of GUIDON-WATCH prompted us to give special care to programming methodology.

In an object-oriented programming language, windows are treated as objects ensuring modularity. In a procedural language, such as a LISP dialect without an object-oriented programming facility, the programmer must take more pains to write well-structured, event-driven display programs. A careless design can make it very difficult to find the code that creates a window, defines its properties, and determines how it will respond to events (mouse buttoning, scrolling, repainting, etc.). In GUIDON-WATCH, most procedures are associated with a specific window. Therefore, the code is organized around the windows. Naming conventions were devised so that the window name is a prefix, and meaningful base words are used as a root. For example, the procedure that creates the Taxonomy window is named TAXONOMY.CREATEW. These simple techniques help make a resulting program more structured, modular, and readable. Additionally, the categorization of procedure types serves as high-

level documentation (e.g., the procedure RULE.REPAINT *repaints* the Rule window). In general, each procedure is associated with a specific computation (e.g. creating a window) or responds to a single event only.

A software designer should consider the graphic display and interface at an early stage in design because it can influence other parts of a complex system including decisions regarding knowledge representation and implementation of the system. The explicit representation of diagnostic knowledge in NEOMYCIN made it possible to provide views of the diagnostic reasoning proccess that would not be possible in MYCIN. However, some of the assumptions made in the Interlisp-10 (DEC-2060) implementation of NEOMYCIN are not valid in a multiprocessing, multiple window software system. For example, the code in the earlier implementation assumed that all output was being written to a single display stream. The availability of multiprocessing may allow for more sophisticated programs that require a great deal of modularization in the code. Therefore, multiprocessing and windowing features should be considered at an early stage if they are available.

Future Work

Several features and facilities could be added to GUIDON-WATCH. Some features would primarily benefit system builders, for example, a graphic editor to facilitate knowledge base construction and editing. Using the INTERLISP-D grapher package, which allows graphs to be interactively edited, a member of the NEOMYCIN project is writing a set of functions that update the underlying data structure whenever the graph is edited.

A trace and break package would allow a user to break any knowledge base object (rules, tasks, hypotheses, findings, and relations) or event. The system might print information or cause a break whenever a specified rule, task, or relation is tried, when a rule or relation succeeds (or fails), whenever the system is

determining a certain finding, or testing a hypothesis, when a conclusion is made about a finding or a hypothesis, or when the differential changes. Additionally, the capability to *replay* part or all of a consultation could benefit both developers and students. A break package of this type provides information and control closer to the conceptual level, in which knowledge in a program is represented, rather than at the LISP programming level. Because the windows of GUIDON-WATCH are organized around knowledge structures rather than LISP data strucutures, the program is ideally suited for such a break package.

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 instructional purposes. Interactive pictures can illustrate facts and processes. A small project resulted in an interactive view of the brain that illustrates where several diseases occur. (figure 9) However, implementing graphic displays can be demanding and time-consuming. Most of the time spent in that project was devoted to implementing a simple drawing program. There is a need for more graphics packages that are integrated with AI programming environments to help software developers more easily utilize the full graphics capabilities of available hardware.

Graphics programming issues provide new research areas in AI. We have mentioned the use of graphics for instruction and general interactive knowledge base browsing. However, issues concerning the development of more intelligent user-interfaces are ripe for investigation. These topics include user-modeling, intelligent presentation, and declarative languages for describing graphical interaction. Mackinlay is investigating some of these issues.[17] For GUIDON-WATCH, we decided how to present information and hand-coded it. Instead, Mackinlay's program *reasons* 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.

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 a mesh of procedural code. The use of 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 is not explicit, and it is difficult to modify. Another advantage of using a declarative representation of knowledge is that it can be used in multiple ways. A program that reasons about the presentation of information can potentially explain its reasoning if an explicit and declarative language is used. This can be useful if a program is an intelligent aid for graphic design since a user may want justification for the program's choices.

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 a 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 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 including computer graphics and AI, psychology,

linguistics, communications, education, and graphic design.

Conclusion

GUIDON-WATCH is a prototype graphic interface to a knowledge-based system. The program demonstrates the feasibility of providing both end-users and system designers with a means of quickly browsing and viewing a knowledge-based consultation system. The program illustrates how multiple windows, menus, and a mouse device can provide a great deal of leverage in achieving this goal. Below we summarize the most important principles learned from this effort.

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 be exploited in this effort. 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.

GUIDON-WATCH provides a simple, flexible, consistent and powerful interface. The program uses the mouse buttons in a simple and consistent manner, but still provides a user with the flexibility and power to browse the knowledge base quickly and thoroughly. Furthermore, interactive graphics are used in GUIDON-MANAGE to allow a user to start and control the course of a consultation. The user is able to browse the knowledge base before each question is answered during a consultation.

The use of bold fonts, boxing and graying items, and other

non-textual cues provide a means of maximizing information content and highlighting facts and events in a way that is quickly understandable, but does not overwhelm the user. The use of these simple techniques in the Evidence and Taxonomy windows illustrates their effectiveness.

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. In well-constrained situations it is possible to automatically manage the display and placement of windows, as in GUIDON-WATCH.

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 HERACLES consultation. Subsequent instructional programs may help students explain, debug, and augment their own reasoning processes, using graphic displays like GUIDON-WATCH to compare and contrast alternative solutions to problems.

Acknowledgements

Diane W. Hasling, a staff programmer on the NEOMYCIN project, and Julie T. Richer both provided many helpful suggestions for improving the clarity of this paper.

This research has been supported in part by ONR and ARI Contract N0001̂4-79C-0302. Computational resources have been provided by the SUMEX-AIM facility (NIH grant RR00785).

References

1. Clancey, W.J., Overview of GUIDON," *Journal of Computer-Based Instruction*, Vol. 10, No. 1 & 2, Summer 1983, pp. 8-15, [Also in *The Handbook of Artificial Intelligence*, Volume 2, eds. Barr and Feigenbaum, Kaufmann, Los Altos. Also STAN-CS-93-997, HPP-83-42.]

2. Shortliffe, E., MYCIN: A Rule-Based Computer Program To Advise Physicians Regarding Antimicrobial Therapy Selection," Tech. report HPP-74-2, Stanford University, October 1974.

3. Clancey, W. J. and Letsinger, R., NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, August 1981, pp. 829-836, [Revised version in Clancey and Shortliffe (editors), *Readings in Medical Artificial Intelligence: The First Decade*, Addison-Wesley, 1984]

4. Van Melle, V., A Domain-Independent System that aids in Constructing Knowledge-Based Consultation Programs," Tech. report STAN-CS-80-820, Stanford University, June 1980, [Available as: *System Aids in Constructing Consultation Programs*, UMI Research Press, Ann Arbor, Michigan, 1981.]

5. Bush, V., As We May Think," *Atlantic Monthly*, July 1945, pp. 101-108.

6. Englebart, D., Advanced Intellect-Augmentation Techniques," SRI project 7079, final report, July 1970.

7. Kay, A., Microelectronics and the Personal Computer," *Scientific America*, September 1977, pp. 230-244.

8. Goldberg, A., Educational Uses of a Dynabook," *Computers & Education*, Vol. 3, 1979, pp. 247-266.

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

10. Brown, J.S., Process versus Product--a Perspective on Tools for Communal and Informal Electronic Learning," *Education in the Electronic Age*, July 1983, proceedings of a conference sponsored by the Educational Broadcasting Corporation WNET/Thirteen.

11. Model, M., Monitoring System Behavior in a Complex Computation Environment," Tech. report STAN-CS-79-701, Stanford University, January 1979.

12. Tsuji, S. and Shortliffe, E., Graphical Access to the Knowledge Base of a Medical Consultation System," *Proceedings of AAMSI (American Association for Medical Systems and Informatics) Congress 83*, May 1983, pp. 551-555.

13. Lane, C., Differding, J., and Shortliffe, E., Design of a Graphic Interface for a Medical Expert System," Tech. report KSL-85-15, Stanford University, 1985, Submitted to ACM Siggraph-85

14. Hollan, J. D., Hutchins, E. L., and Weitzman, L., STEAMER: An interactive inspectable simulation-based training system," *The AI Magazine*, Vol. 5, No. 2, 1984, pp. 15-27.

15. 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, .

16. Hasling, D. W., Clancey. W. J., and Rennels, G., Strategic Explanations for a Diagnostic Consultation System," *International Journal of Man-Machine Studies*, Vol. 20, 1984, pp. 3-19.

17. Mackinlay, J., Intelligent Presentation: The Generation Problem for User Interfaces," Tech. report HPP-83-34, Stanford University, March 1983.

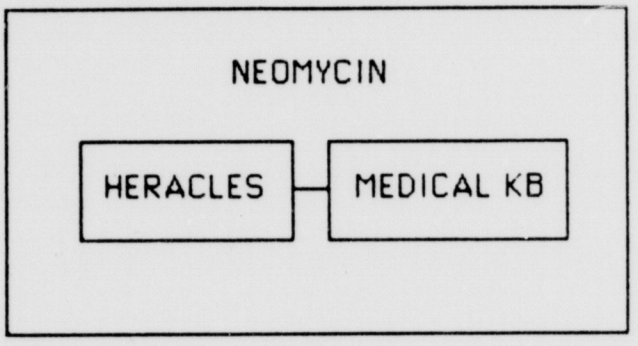


Figure 1. NEOMYCIN consists of HERACLES and a knowledge base of medical facts and rules.

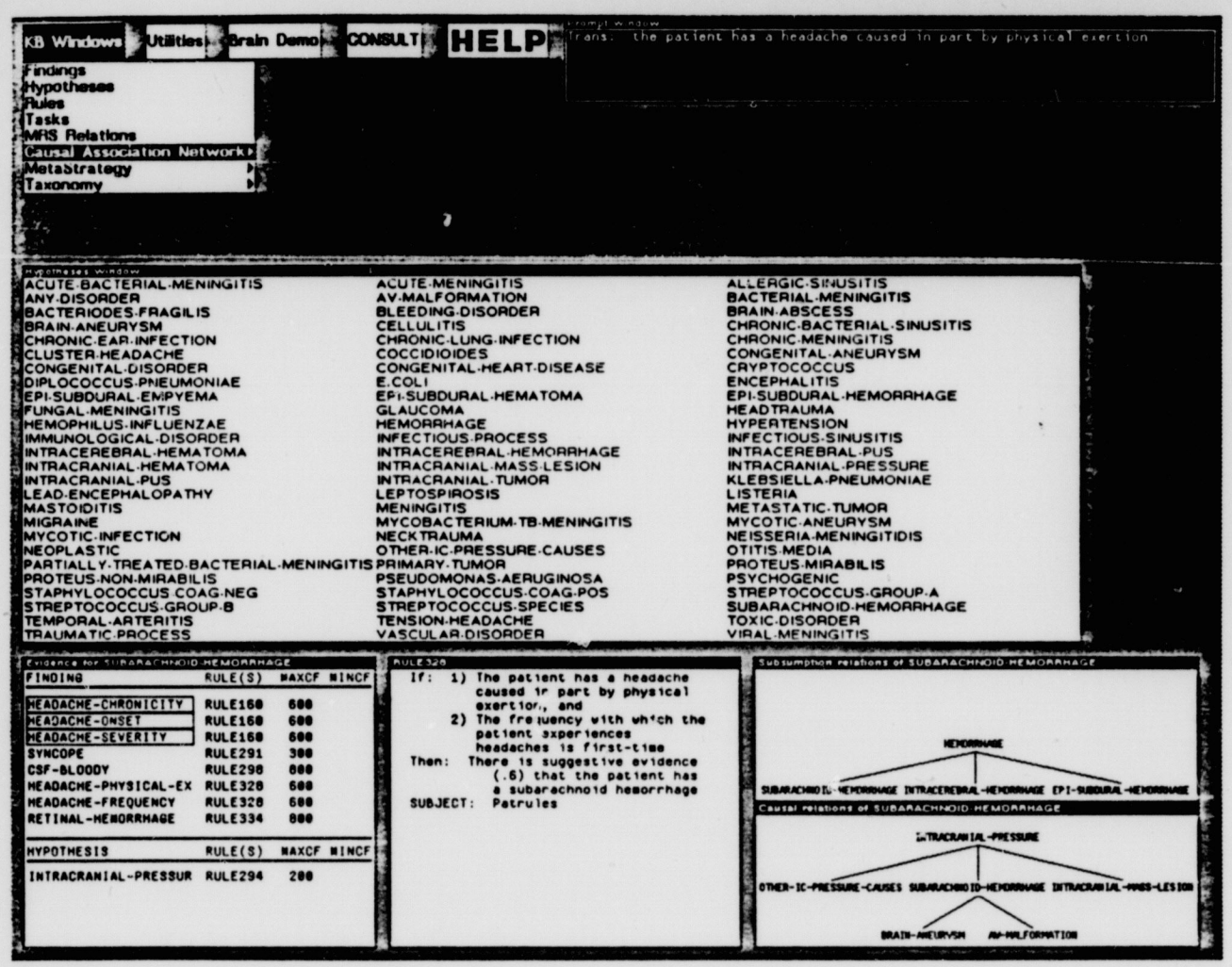
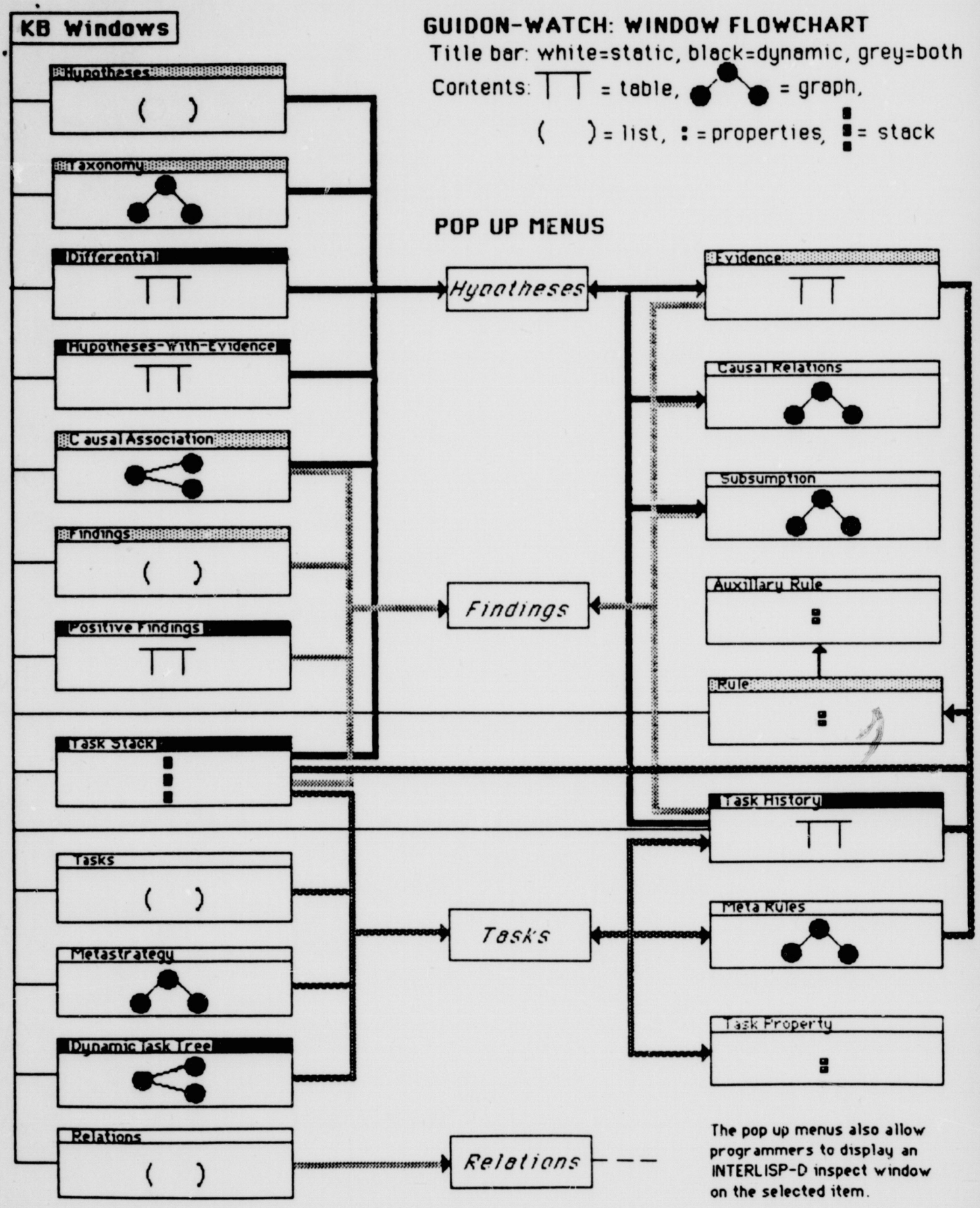


Figure 2. The user has selected the *KB Windows* pull down menu and has moved the mouse cursor over the item *Causal Association Network* with the left mouse button depressed. If the user releases the mouse button now, the Causal Assocation window would be displayed. Some items have a small arrow on their right side; this indicates the user can display a submenu for that item. The user has displayed several windows on the screen already. The Hypotheses window shows all the hypotheses represented in the NEOMYCIN knowledge base. The other windows are described in the text. Note that the user had selected "HEADACHE-PHYSICAL-EX" in the Evidence window to get an English translation printed in the Prompt window (top right of screen). In general, the Prompt window is used to print messages to the user.


KB Windows
GUIDON-WATCH: WINDOW FLOWCHART
Title bar: white=static, black=dynamic, grey=both
Contents: = table, = graph,
( ) = list, : = properties, = stack
POP UP MENUS
Hypotheses
( )
Taxonomy
Differential
Hypotheses-With-Evidence
Causal Association
Findings
( )
Positive Findings
Task Stack
Tasks
( )
Metastrategy
Dynamic Task Tree
Relations
( )
Hypotheses
Findings
Tasks
Relations
Evidence
Causal Relations
Subsumption
Auxillary Rule
Rule
Task History
Meta Rules
Task Property
The pop up menus also allow programmers to display an INTERLISP-D inspect window on the selected item.

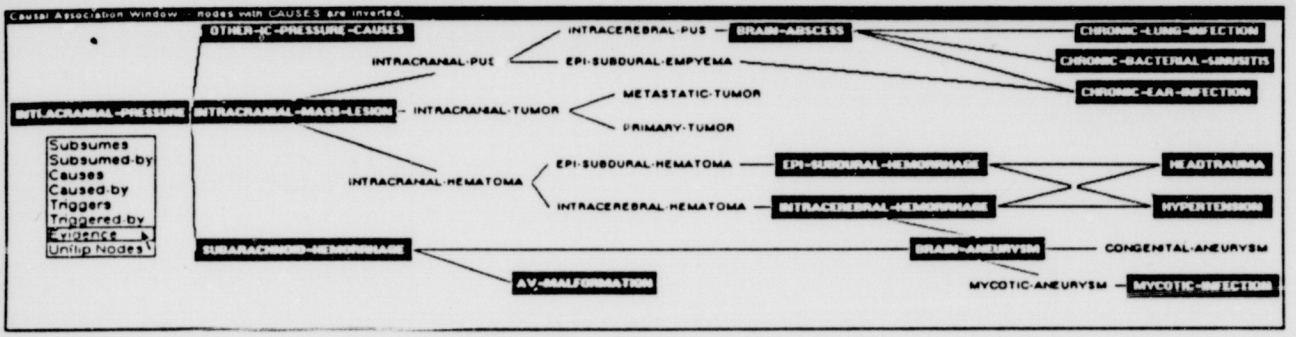


Figure 4. The Causal Association window has a pop up menu associated with it that is displayed when the user buttons away from the nodes with the middle button. Previously the user chose to see which nodes have a *Causes* property; those nodes are inverted in the picture above.

Figure 3. Most of the windows in GUIDON-WATCH are shown in the diagram above. The *KB Windows* pull down menu allows a user to display most of these windows. This is shown by drawing a line from the left side of the window to the vertical line that is attached to the box containing "KB Windows." There are lines connecting the right side of each window to the pop up menu(s) that can be displayed in that window. Some windows also contain rules and can directly display the Rule window (as shown by a line from the window to the Rule Window). Some windows are displayed from pop up menus only. These windows can also have pop up menus; therefore, the lines connecting the pop up menus and the windows on the right side of the figure can be bi-directional.

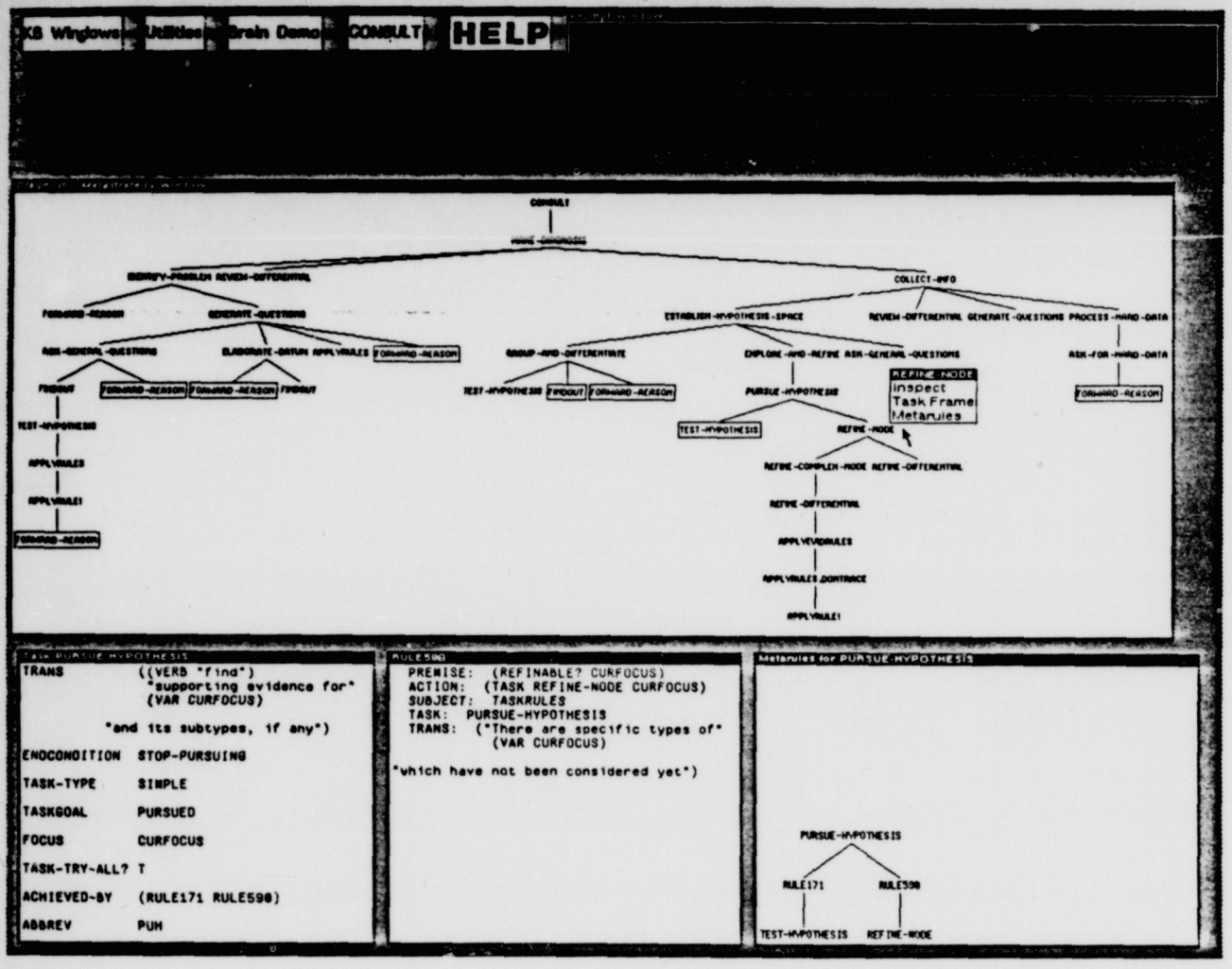


Figure 5. The user has selected "REFINE-NODE" in the Metastrategy window and a pop up menu is displayed.

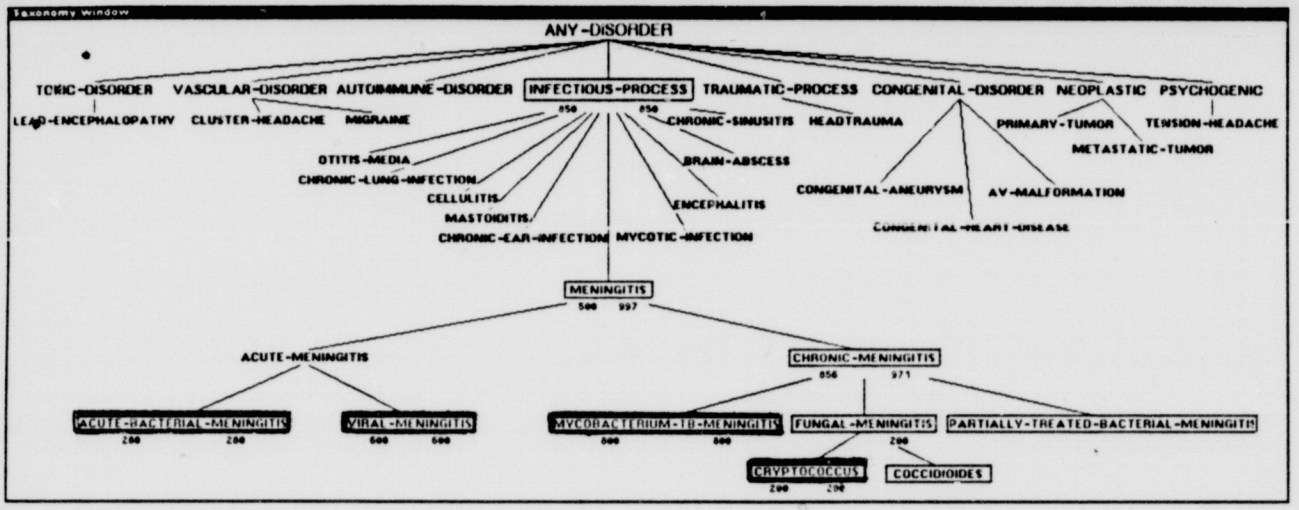


Figure 6. During a consultation dynamic information is displayed in the Taxonomy window. Dark boxes appear around nodes that are on the differential; the lighter boxes indicate disorders that were previously on the differential. Certainty factors are printed under the nodes.

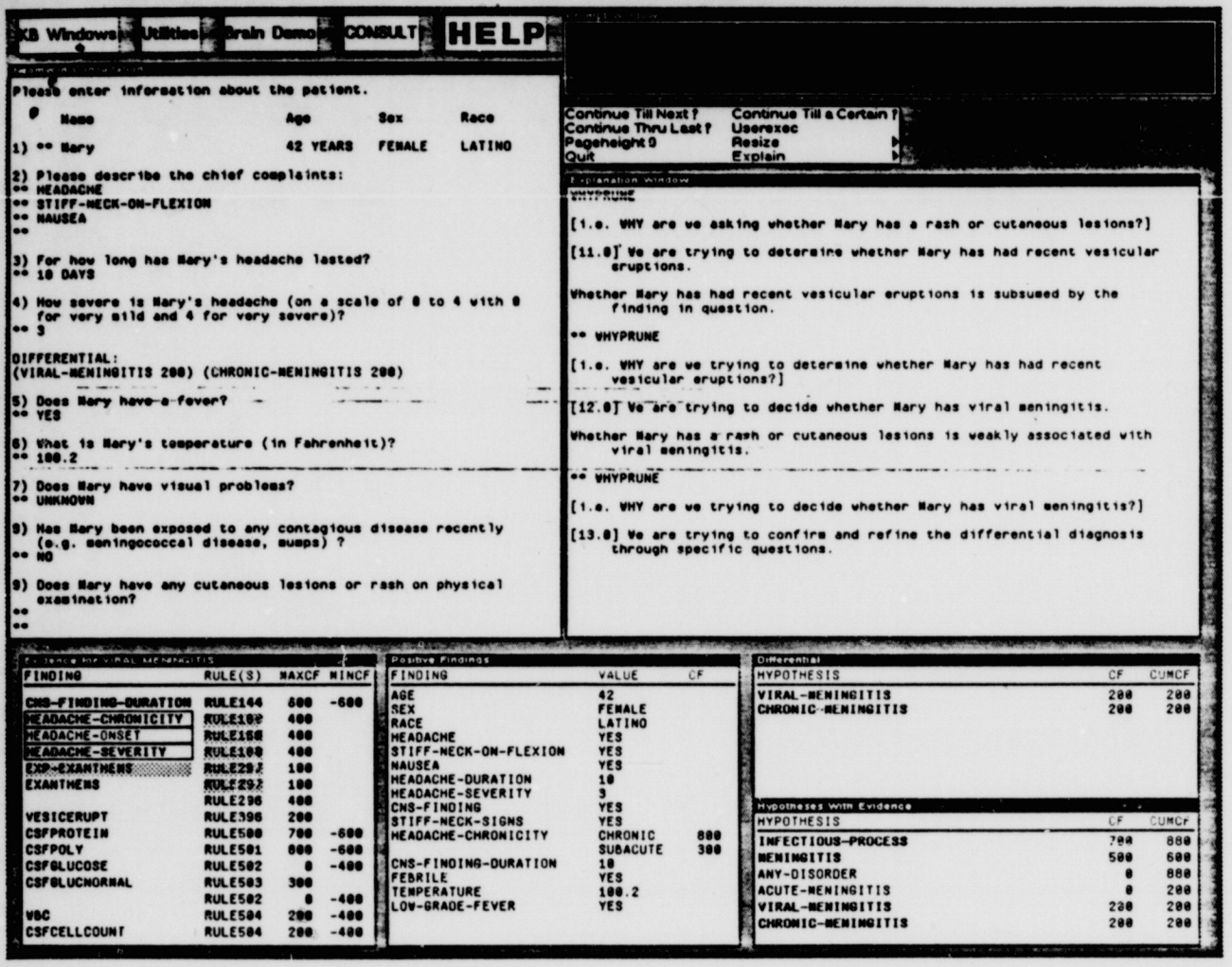


Figure 7. The windows above include the Consultation typescript, the Explanation window, and several other windows displaying dynamic information. The menu that is attached to the typescript window allow the user to continue the consultation, request explanations, etc. While this menu is displayed the user can use the mouse to select any menu item or knowledge base object. The menu is closed while the program is "reasoning"; at that time, the mouse cursor also changes to a "CONSULT" icon.

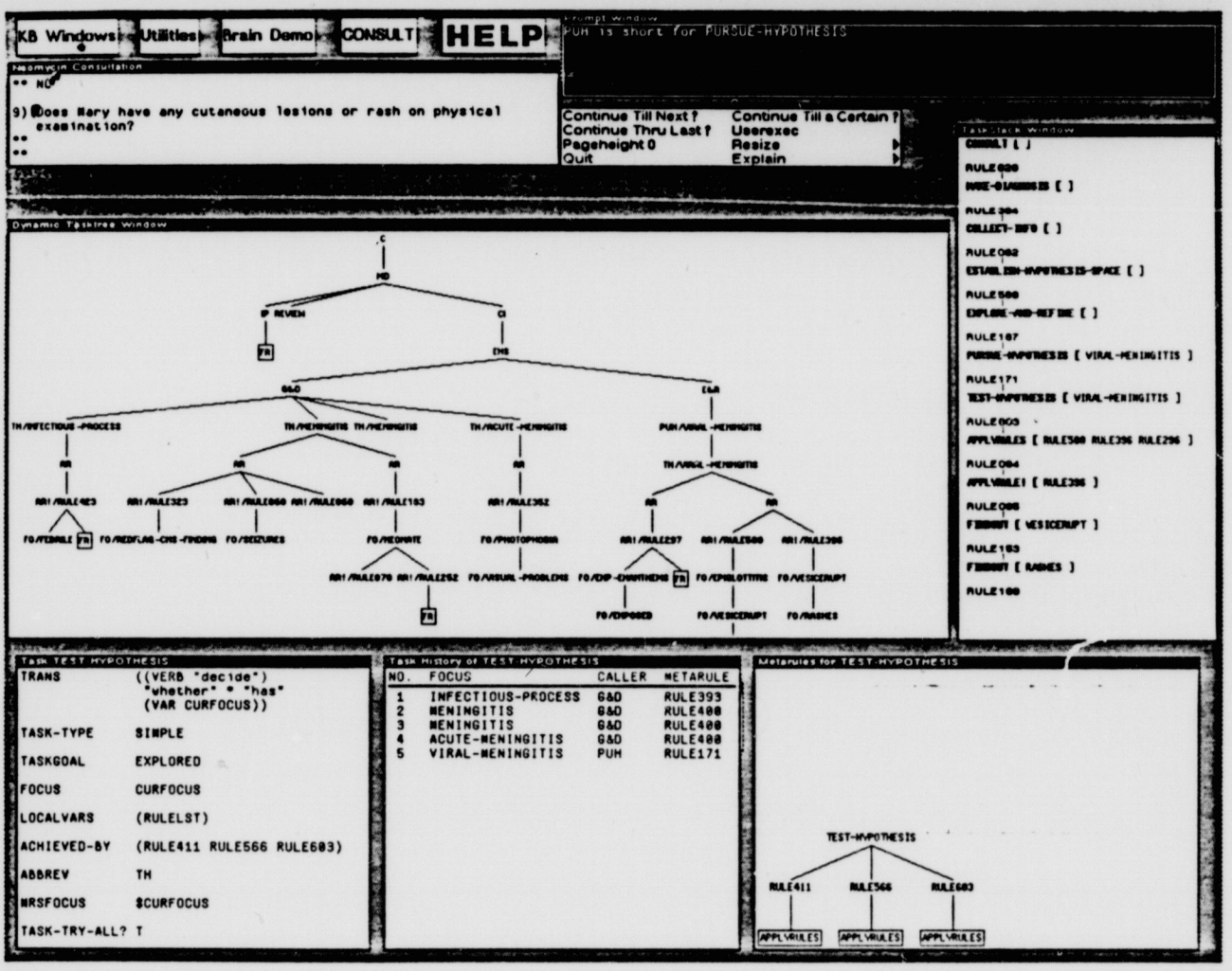


Figure 8. The dynamic tasks windows are shown above. To save space, task names may be abbreviated. The full name can be displayed in the Prompt window by selecting the abbreviated item; a pop up menu will also be displayed.

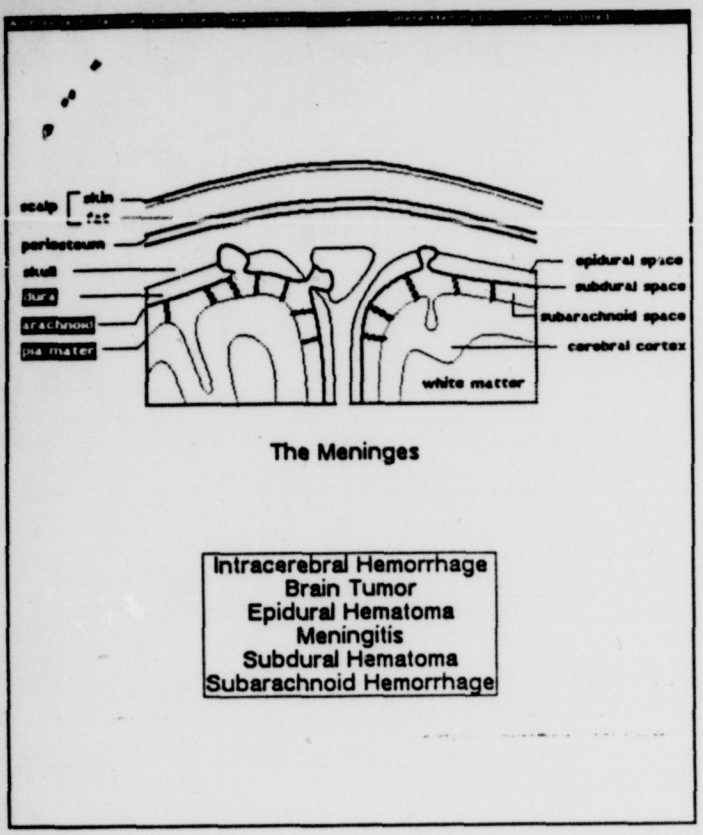


Figure 9. A simple drawing program was used to generate the picture show above. If the user selects a disease in the menu (bottom part of the window), the labels that correspond to the parts of the brain where the disease can occur are inverted. As shown above, meningitis can occur in the dura, arachnoid, and pia mater.