ARI Research Note 2001-11

Mentor: Dialog Agent System for Mentoring and Conversational Role-playing

William R. Murray and Michelle Sams
Teknowledge Corporation

William H. DeSmedt and Donald Loritz
Amber Consortium

20010808 092

Research and Advanced Concepts Office

August 2001

U.S. Army Research Institute
for the Behavioral and Social Sciences

U.S. Army Research Institute
for the Behavioral and Social Sciences

A Directorate of the U.S. Total Army Personnel Command

EDGAR M. JOHNSON
Director

Research accomplished under contract
for the Department of the Army

Micro Analysis and Design

Technical review by

Jonathan Kaplan

NOTICES

REPORT DOCUMENTATION PAGE

1. REPORT DATE (dd-mm-yy) August 2001	2. REPORT TYPE Final	3. DATES COVERED (from. . . to) 05-01-98 to 06-30-01
4. TITLE AND SUBTITLE Mentor: Dialog Agent System for Mentoring and Conversational Role-playing		5a. CONTRACT OR GRANT NUMBER DASW01-98-C-0006
		5b. PROGRAM ELEMENT NUMBER 2O665502M770
6. AUTHOR(S) William R. Murray and Michelle Sams (Teknowledge Corporation), William H. DeSmedt and Donald Loritz (Amber Consortium)		5c. PROJECT NUMBER
		5d. TASK NUMBER
		5e. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Micro Analysis and Design, Inc – 4900 Pearl E. Circle, #201E, Boulder, CO 80301 Teknowledge –1810 Embarcadero Road, Palo Alto CA 94303 The Amber Consortium, Inc – 1010 Tower Rd, Milford, PA 18337		8. PERFORMING ORGANIZATION REPORT NUMBER MA&D 7031
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Institute for the Behavioral and Social Sciences 5001 Eisenhower Avenue Alexandria, VA 22333-5600		10. MONITOR ACRONYM ARI
		11. MONITOR REPORT NUMBER Research Note 2001-11

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT *(Maximum 200 words)*:

The goal of this SBIR program was to provide authorable, dialog-enabled agents for tutoring and performance support systems. Users interact with agents who carry out strategies and goals and can engage in mixed-initiative dialog via a natural language understanding and generation system. Non-programmers can author new domains and scenarios and create new dialog agents. The dialog system is authorable by non-computational linguists. The system has two types of agents, Mentor agents and Conversational agents. The Mentor agent is a simulated subject matter expert (SME) that provides troubleshooting and problem solving advice. Mentor engages in a dialogue with trainees, helping them solve problems by taking them through logical courses of action and asking and answering domain-specific questions. Conversational agents are used for role-playing scenarios. The only real difference between the two agents is that Conversational agents do not have specific problem solving strategies. Both Mentors and Conversational agents have domain specific knowledge and access to a common sense knowledge base. This report describes the capabilities and limitations of results of this Phase II effort.

15. SUBJECT TERMS natural language understanding, mixed-initiative dialog, mentor, intelligent agents, knowledge base, conversational agents, SBIR

SECURITY CLASSIFICATION OF			19. LIMITATION OF ABSTRACT	20. NUMBER OF PAGES	21. RESPONSIBLE PERSON (Name and Telephone Number)
16. REPORT Unclassified	17. ABSTRACT Unclassified	18. THIS PAGE Unclassified	Unlimited	50	Dr. Jonathan Kaplan, ARI 703-617-8828

CONTRACT NO. DASW01-98-C-006

Final Report
Mentor: Dialog Agent System for Mentoring and Conversational Role-playing

William R. Murray, Michelle Sams
Teknowledge Corporation

William H. DeSmedt, Donald Loritz
Amber Consortium

June 2001

United States Army Research Institute for the Behavioral and Social Sciences

Approved for public release; distribution is unlimited.

MENTOR: DIALOG AGENT SYSTEM FOR MENTORING AND CONVERSATIONAL ROLE-PLAYING

MENTOR: DIALOG AGENT SYSTEM FOR MENTORING AND CONVERSATIONAL ROLE-PLAYING

INTRODUCTION

Army Need

With high turnover and increasingly complex tasks and equipment, the Army has a need for more efficient training and performance support. A computer-based performance support system can provide assistance over the Internet or in stand-alone capability, when and where needed. A subject matter expert's time could then be better leveraged for tutoring more difficult problems better solved in person.

Most electronic performance support systems (EPSS) are duplications of manuals, with some hyperlinks. While this electronic delivery provides self-paced assistance, it does not adapt to the trainee and is not truly interactive. Trainees must still hunt for the answers to the problems, often through nested menus. There is no opportunity to ask questions or get focused assistance.

What is needed is a generic, easily authored performance support system that can generate coaching interactions in natural language.

Project Description

The goal of this program is to provide authorable, dialog-enabled agents for performance support. Trainees converse as they would in normal English discourse by asking questions, giving answers, and making comments or requests. The dialog agents respond to user's comments and queries, and generate questions and comments of their own. They do not simply retrieve dialog responses. They model an understanding of the user's natural language (NL) input and then generate an appropriate natural language response from knowledge bases.

The system has two types of agents, Mentor agents and Conversational agents. The Mentor agent is a simulated subject matter expert (SME) that provides troubleshooting and problem solving advice. Mentor engages in a dialogue with trainees, helping them solve problems by taking them through logical courses of action and asking and answering domain-specific questions. Conversational agents are used for role-playing scenarios as they can carry on a mixed-initiative dialog. The only real difference between the two agents is that Conversational agents do not have specific problem solving strategies. Both Mentors and Conversational agents have domain specific knowledge and access to a common sense knowledge base. Both types of agents have dialog management strategies, such as tracking dialog and turn-taking rules. We will refer to them as Dialog Agents or the Dialog Agent System (DAS) when addressing features in common to both types of agents.

Non-programmers can create new Dialog Agents, assigning them names and personalities. Without any programming required, authors can add new facts to an agent's domain specific knowledge base and add terms or rules to the common sense knowledge base. Non-programmers can easily add new terms to the dialog system. These are very innovative authoring features, as other systems typically require a knowledge representation engineer to add new facts and a computational linguist and/or an ontologist to add new terms.

Results

The major advances in this project include the development of a web-based agent capable of natural language dialog (not pattern-matching or modified context-free grammar approaches, but a full-fledged natural language understanding and generation capability) and an authorable dialog system (knowledge bases, lexicon, ontology).

Although promising, the results of this project indicate that much more remains to be done. Because conversing in English is natural and easy for humans, expectations may be unrealistic about what can actually be accomplished in a dialog software system. As with many previous NL systems, the Dialog Agent System (DAS) demonstrates a capability of handling both syntactic and semantic variability but has some limitations that are detailed later in this report. The DAS ontology allows it to recognize similar concepts and perform some simple deductions about the subject matter. However, this project demonstrates that an extensive common-sense knowledge base and robust reasoning capability is required to support a flexible conversational and mentoring dialog.

The majority of the funds in this project were focused on the development of the dialog system, and thus we employed a basic method for the Mentor agent to give procedural or problem-solving advice. Rather than using expert systems or other artificial intelligence techniques, Mentor uses a simple state-space path approach called an *effective problem space*. Mentor's strategy is based on a sequence of observations or tests (a problem tree), and the next step is suggested based on responses by the user (such as, results of a test or answer to a query).

This project also created the rudiments for personality to be displayed in an agent's conversation. The particular challenge of this task was to create a method for this variable to be authored and generated in conversation on the fly. We only made small steps in this direction in this project.

While the system provides the capability to create up to six different dialog agents, it does not provide the capability to converse with more than one agent at a time. The particular challenge of multiple agents in a single session is to develop the capability for each agent to process what is said by the human user *and* by other dialog agents.

In conclusion, the Dialog Agent System succeeded in providing a natural language mixed-initiative conversational system that is fairly easy to author. However there are some capability limitations that will be discussed.

DIALOG AGENT SYSTEM FUNCTIONALITY

In this section we will briefly describe some of the system's chief functions along with some examples. We will address the dialog goals in mixed-initiative conversation, Conversational agents and Mentor agents, and common-sense reasoning. We will describe the technical approach and authoring in more detail in later sections.

MIXED-INITIATIVE CONVERSATION

The Dialog Agent System supports mixed-initiative dialog by following goals. These goals are different for Mentor agents and Conversational agents.

The primary goal of a Mentor agent is to traverse the nodes in a problem-solving tree until a terminal node is reached (a conclusion). A Mentor agent will generate dialog based on the type of currently active node, such as ask questions or suggest actions or make recommendations. Trainees can ask questions of the Mentor agent at any time in a *side-bar conversation.* The Mentor agent temporarily pauses its problem-solving strategy and generates a response to the trainee query. The response is created from relevant facts in its own knowledge base or in the common sense knowledge base. If the trainee stops asking questions, makes a comment, or makes a query to which the Mentor can find no relevant facts, then it resumes the dialog at the last unanswered problem node. After completing a problem tree, the user can still carry on a conversation with the Mentor agent about any factoids that reside in its own knowledge base or common sense knowledge base.

A Conversational Agent answers queries posed by the user based on its own knowledge base and common sense knowledge base. It uses a simple strategy to initiate questions. When the user is not asking a question, or makes a comment or query to which the Conversational agent can find no relevant facts, it poses a question to the user based on the sequential order in which they were authored.

CONVERSATIONAL AGENTS

Conversational agents (CA) are agents with different world-views (belief systems). An author creates world views by entering facts to the character's own knowledge base. Conversational agents also have access to the common sense knowledge base.

The facts in a character's knowledge base could look like the following:

You are an astronomer.
You are a Greek.
You want to discover new stars.

A user can ask questions about the subject matter domain or common sense knowledge base with a conversational character. Conversational agents have basic discourse goals that prompt them to ask questions of the user. The basic dialog strategy is to pose a question to the user when the user has stopped asking questions or makes a statement that requires no response from the CA. Questions are posed based on the sequential order in which they were authored.

Student: Is the earth flat?

Copernicus: No, the earth is not flat.

MENTOR AGENTS

Although Conversational agents can *talk about* a subject matter domain they do not have problem solving strategies. Additional knowledge, in the form of an effective problem space, is provided for this kind of support for Mentor agents. A Mentor agent can provide coaching assistance for the user, such as in basic troubleshooting or problem solving. In the case of troubleshooting, the effective problem space plays a similar role to conventional fault trees found in maintenance manuals, suggesting tests to perform and actions to take based on user feedback of test results. The effective problem space also provides additional information about what can be concluded as the results of tests (e.g., diagnosis, hypotheses). This additional information can be used by the Mentor agent to explain its reasoning and actions.

Mentor is not an intelligent tutoring system (ITS) as it lacks the student model and overall pedagogical control that are key components of an ITS. However, as Mentor is intended to generate coaching interactions, converse in English, and be easily authored, it provides capabilities that are far more flexible than typical computer-based training (CBT) systems. It also provides subject matter expertise, which *is* one of the other key defining characteristics of an intelligent tutoring system. So there is some overlap in the capabilities but we would still hesitate to call Mentor an intelligent tutoring system as it lacks the other two components typically found in an ITS.

A target domain was selected to demonstrate the capabilities of Mentor-- solving boot problems for SUN workstations. An effective problem space for SUN workstation boot problems was modeled and includes typical "bugs" that were culled from user documentation and subject matter reviews. Many of the dialogs and conversations in this report are from the Sun workstation boot problem, but some other domains are also included for illustration.

Here is an example of a conversation with a Mentor agent for a computer problem:

Lt Ambroid: Can other computers talk to your computer?

Ima Trainee: Yes, they can.

Lt Ambroid: Can your computer talk to other computers?

Ima Trainee: No, they cannot.

Lt Ambroid: There may be a problem in the host's table. Please check the host's table to make sure that the IP address of your computer is correct. If incorrect, other machines could not talk to yours.

The Mentor agent's primary conversational goal is to work through the effective problem space with the trainee, asking questions and suggesting actions. However, the trainee can ask a question at any time. Mentor will temporarily pause its progression through the problem space, and look for information relevant to the question in its own knowledge base and in the common knowledge base. If it can find relevant information, then it will formulate a response. Mentor will continue to answer questions as long as the trainee poses them. If appropriate information cannot be found or if the student ceases to ask questions, then Mentor resumes its progression through the problem space by repeating the last unanswered Mentor question or suggestion action. We call these types of dialog, side-bar conversations. Side-bar conversations may or may not be central to the task at hand, and cause the agent to shift its dialog goals temporarily.

Here's an example of a side-bar conversation, where the Mentor agent temporarily stops its progression through the problem space to answer the trainee's question about a SIMM, and then resumes the problem solving strategy:

> Lt Ambroid: Can other computers talk to your computer?
>
> Ima Trainee: Yes, they can.
>
> Lt Ambroid: Can your computer talk to other computers?
>
> Ima Trainee: What is a SIMM? {side-bar question}
>
> Lt Ambroid: A SIMM is a kind of memory module. {side-bar response}
>
> Lt Ambroid: Can your computer talk to other computers? {resumes problem strategy}
>
> Ima Trainee: No, they cannot.
>
> Lt Ambroid: There may be a problem in the hosts table. Please check the hosts table to make sure that the IP address of your computer is correct. If incorrect, other machines could not talk to yours.

After a Mentor agent draws to a conclusion in the problem space, it can continue to carry on a dialog with the trainee. It will answer any questions with relevant information from its own knowledge base and the common sense knowledge base. It will also ask questions that have been authored outside the effective problem space domain.

> ...
>
> Lt Ambroid: There may be a problem in the hosts table. Please check the hosts table to make sure that the IP address of your computer is correct. If incorrect, other machines could not talk to yours.
>
> Lt Ambroid: Is there anything else you would like to know?
>
> Ima Trainee: Who are you?
>
> Lt Ambroid: I am Lt Ambroid.
>
> Ima Trainee: sorry I forgot your name
>
> Lt Ambroid: Do you like the Army? [asks the first non-problem domain question previously authored, because problem space was concluded and Ima Trainee did not pose a further question]
>
> Ima Trainee: Yes, I do.

COMMON-SENSE KNOWLEDGE BASE AND REASONING

Each Dialog agent (Conversational agent and Mentor agent) also inherits global knowledge from a central knowledge base, called Common Knowledge. Examples of the factoids that might appear in the Common Knowledge base are:

All dogs have four legs.

All dogs like to chase cats.

Facts such as *All dogs are mammals* do not need to be specifically authored as they are implicit by their location in the ontolexicon (more on this later).

If there is a conflict between a fact in the agent's specific knowledge base and the common sense knowledge base, the agent will 'believe' the fact in its own knowledge base. These individual belief systems are useful for creating characters with different experiences and knowledge.

For example, an author can create historical characters with different beliefs. If the common sense knowledge base states that the world is round, then all characters will believe that the earth is round unless a contrary statement is entered into the character's own knowledge base.

The dialog system can make simple deductions about subject matter propositions and propositions about themselves. For example, given the following rules in Common Knowledge:

All lawyers are rich.

All rich people have large houses.

and the following fact in the belief set of a the specific Conversational agent, Mr. Darrow:

You are a lawyer.

The system can infer from the common knowledge base that Darrow has a large house and is rich, even though it was not explicitly authored in that specific character's knowledge base.

Questioner: Do you have a large house?

Mr. Darrow: Of course I have a large house.

Questioner: Are you rich?

Mr. Darrow: I am rich.

Actually, the capability to resolve conflicts is broader than just specific vs. common knowledge KBs. It also supports such conflict resolution *within* a single Kb. For instance, if the common knowledge KB contains two factoids:

1. all mammals have four legs.

2. All bats have two legs.

...then the DAS will permit [2] to override [1] when asked about the number of leg bats have. The logic, in other words, falls under the heading of general non-monotonic reasoning.

AUTHORING

In this section we provide an overview of how the Dialog Agent system is authored. Note that authoring and running the Dialog Agents are separate processes. New terms, factoids, rules or changes to an effective problem space cannot be provided while in midst of a dialog.

CREATING A DIALOG AGENT

The authoring system allows creation of a new dialog agent from scratch or editing of an existing agent. Currently the system has six character slots available. Any of these slots can be used to create Conversational Agents or Mentor Agents. However, users can only have a dialog with one character at a time.

In the creation of a new character, the author selects the identity elements: title (Dr., Lt.), the first and last name, and the three personality attributes (gender, helpfulness, and manner). The options for the three character attributes are selected via drop-down lists.

Selection of helpfulness and politeness levels affects the dialog characteristics of the agents. Toward the casual-rude end of the spectrum, the DAS will being assuming it's on a first-name basis with its collocutor, or even start insulting them.

1) Gender—the gender of the agent: male, female, or neuter.

2) Helpfulness—how cooperative the agent is in answering questions: indifferent, cooperative, uncooperative.

3) Manner—how polite the agent is to the user in addressing them: obsequious, polite, normal, casual, impolite, rude.

ADDING KNOWLEDGE (FACTS)

The next authoring screen allows facts to be added to the character's own belief set (knowledge base). A fact is entered as a complete sentence. Facts which describe the character itself are given as directions: You believe x, You are y, You like z.

You are an artist.

After each sentence is entered as a fact, it is proofed by the dialog system. The proofing process checks to make sure it 'understands' the words and the sentence. It then transforms the sentence to its own format for a factoid. For example, the factoid transform of the previous fact is: You are now an artist. [The representation that is actually saved 'internally' looks like a case-frame whose slots are populated with numeric concept-encodings.]

If the dialog system cannot parse the sentences or the questions, or if unknown terms are used, then the dialog system will italicize the unknown text. Sometimes rephrasing the sentence or question will result in acceptance by the dialog system. Individual unknown terms need to be entered into the dialog system (this authoring process is described later).

If we want all Conversational agents and Mentor agents to have access to the same knowledge (facts), then these facts are added to the Common Knowledge base.

All diodes have two leads.
The sum of the currents flowing into a node is always 0.
The earth is round.

Providing these facts in the Common Knowledge base does not allow Mentor agents to dynamically incorporate them into new problem solving strategies, but can be drawn upon to formulate responses to trainee's side-bar questions. To employ problem-solving strategies, an effective problem space needs to be authored for the domain.

AUTHORING A MENTOR

To create a Mentor agent, the author creates a new character and assigns it a name and personality attributes. One can then add general or domain specific facts to its knowledge base or to the common sense knowledge base.

In addition, the author must create an effective problem space (EPS). The EPS simulates knowledge and strategies about *what to do next* in a procedure or in a simple problem-solving domain. A Mentor agent uses the EPS as the strategy for its dialog interaction with the trainee. The EPS can model any problem solving and troubleshooting domains, as long as they can be represented as procedural reasoning in a decision tree.

The EPS is a tree composed of a several kinds of nodes. Nodes represent actions for the trainee to take, or tests/questions or test results/answers to questions, or conclusions. Mentor reasons by following a path from the root of the tree through alternative paths trying to reach a terminal node leading to a conclusion or recommendation.

To create an EPS, the author first provides a title for the problem space, such as "Troubleshooting SUN workstation boot problems". The author creates the decision tree by dragging and dropping the different kinds of nodes onto a display space. Prompts are provided for some text entries. [see Appendix for examples of the authoring screens]

The authoring system also has internal consistency checks about the ordering of nodes. For example, a Test node can be placed below any node except another Test node (as every test node should be followed by Test Results).

After an EPS tree is created, the text added for each node is *proofed* (checked to see if the dialog system understands it). The system semi-automates the proofing process by pulling in each node narrative into the dialog authoring system one node at a time. After an EPS has been successfully proofed then the Mentor agent can provide dialog-based problem-solving strategies.

AUTHORING CONVERSATIONAL AGENTS

To create a Conversational agent, the author creates a new character and assigns it a name and personality attributes. One can then add general or domain specific facts to its knowledge base or to the common sense knowledge base.

Questions that the Conversational agent might pose to the human dialog partner are not authored in question format, as one might suspect, e.g., in a form like "When did you join the Army?" Instead, they are authored as factoids such as "You joined the Army" for the previous example. Any factoid entered into the User KB are flagged with 0 certainty, rendering them as questions for the purposes of language generation.

EXTENDING THE LEXICON AND ONTOLOGY

New topic domains will likely have numerous terms particular to that domain, along with domain-specific facts. For an electronics domain we would need to define semiconductor, resistor, transistor, diode, capacitor, and all the other terms that we would expect to encounter in that domain. Next we would want to add facts that are the background knowledge for the domain. For example, a transistor always has three leads.

New terms can be entered as individual words first, or the author can choose to author the facts to the knowledge base first. When facts are added, the dialog system performs a check on all terms to see if they are present. If they are not, then it will italicize the unknown term and provide prompts for authoring new terms. If the author suspects that there are many new terms, it will be easier to enter the new terms individually first, and then enter the associated facts.

To add a new term, the author selects its lexical part of speech from a list of alternatives (noun, verb, adjective, adverb). Then the term must be placed under an appropriate category in the ontology. The ontology is a hierarchy of concepts – a large tree-like structure. Each branch and leaf of the tree inherits characteristics of the preceding concept. [see Appendix for examples of authoring screens]

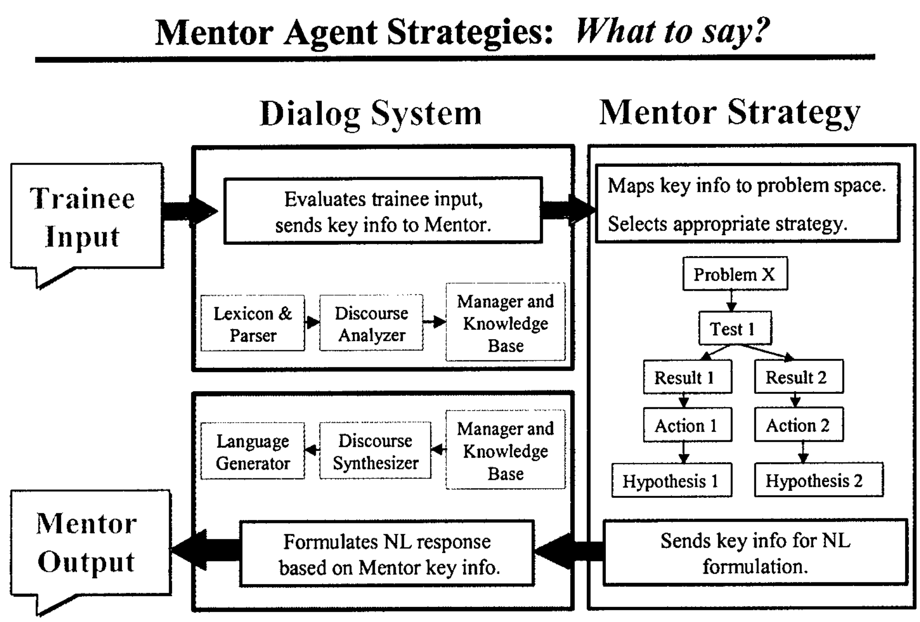
To find the proper category the author searches to see if a relevant concept is already defined. In this case, if a search for "semiconductor" shows that this specific term is not in the ontology but the term "conductor" is present, then "semiconductor" can be placed in the ontology immediately under "conductor".

Once a place in the ontology is determined, then the author needs to specify the relationship of the new term to the existing term next higher up in the ontology. Relationship types are: a kind of, a synonym, or a specific individual.

For example, a "semiconductor" is defined *as a kind* of "conductor". If "Diode 1N034"may be a particular diode in a schematic Diode 1N034 would be entered as an as *a specific individual* under the class "diode" in the ontology.

If there is no logical location in the ontology for a new term, then the author can create a new branch from which to 'hang' the new term.

The Mentor system consists of two main entities. The dialog system and conversational agent facility were created by Amber Consortium. The Mentor agent and overall system control were created by Teknowledge Corporation. The basic concept was to have the dialog system handle the NL interface and the Mentor agent would determine what to do next. Each component communicates with the other.



The Tek Mentor agent controls the strategy for solving problems or troubleshooting. The Mentor agent is divided into an application server and the EPS GUI. The latter is only running when an EPS is being edited. The application server handles the traversal of an EPS. The Mentor agent also controls all communication between the Tek and Amber agents.

The Dialog system has:

1) The Amber server—an HTTP server that has CGI scripts to intercept messages to the Amber system

2) EngPars—a parser and lexicon for English

3) MetaLang—the ontology and knowledge representation system that handles the ontolexicon and reasoning from rules in Common Knowledge and a character's belief set

4) Dramatist—an authoring system for characters' personalities and beliefs

There is also a launcher to start both the Amber dialog agent and the Tek Mentor agents initially, but this is a simple DOS batch file.

The different parts of the Amber dialog system communicate via shared memory in a DLL (dynamic linked library). The different parts of the Teknowledge Mentor agent communicate via messages sent over sockets. Finally, the Amber agent and the Tek agent themselves communicate with each other via HTML HTTP – messages sent to and received from the Amber server. These messages use a special protocol based on KQML (Knowledge Query Manipulation Language), a commonly used inter-agent communication language.

An HTML front-end is provided for the overall Mentor system GUI. Different paths through that GUI lead to different Mentor capabilities (conversing with a conversation agent, conversing with a mentor, editing a character's belief set, editing a mentor agent's EPS, and extending the ontolexicon). JavaScript is also used in the HTML pages for form-validation, to list options (e.g., different agents that can be talked to), and to pass information (via cookies) to the EPS authoring tool.

PROBLEM-SOLVING STRATEGIES: CAPABILITIES AND CURRENT LIMITATIONS

CAPABILITIES

The Mentor agent uses an effective problem space (EPS) as the base strategy for coaching trainees. Traversing starts at the root node which labels the kinds of problems addressed by this problem space (e.g., "SUN workstation will not boot"). When a test node is encountered, the Mentor Agent sends a message to the dialog system to ask the trainee this question. The dialog system then rephrases the query and adds its character traits to the dialog. Then the dialog system sends a message back to the Mentor Agent to display this query with this prompt.

Expected answers or test results are represented by test result nodes below the query (test) node. Each test result node represents a different acceptable answer. When the student replies to the question, the answer is sent to the dialog system in a message from the Mentor Agent that says essentially, "Which answer did the student give?" The dialog system then evaluates semantic equivalence of the student's answer to one of the alternatives. It returns the text of the node semantically matched or else it returns a message that no semantic match was found. In the latter case, the question is asked again.

Once a path is determined by the student's answer, the next node on the path is examined. An Inference node results in a statement by the Mentor agent, such as "At this point I believe that the power fuse has burnt out." An Action node results in a directive to the student, typically leading to a repair or enabling another test, such as "Please open the back of the computer now."

At the end of a path a conclusion should be reached or a recommendation for further assistance given. An example of a conclusion would be, "The IP address for the server is incorrect. Please change it in the etc/hosts table to the correct IP address." An example where the fault cannot be diagnosed given the EPS might be, "We have ruled out obvious faults in the power supply, please ask your instructor for further assistance. The diodes and transformer also appear to beOK given our tests so far."

LIMITATIONS

Alan Lesgold coined the term "effective problem space" to describe the annotated fault-tree simulation used for instruction in the Sherlock intelligent tutoring system. The annotations include explanations for why actions are performed or not performed. They could also include what fault hypotheses are consistent with observations taken so far and which actions make sense to take next, and which are optimal. Thus each state in the effective problem, represented by a tree node, pre-stores information that could be obtained dynamically if an AI-based system or human expert were available for consultation at that time.

The effective problem space can be viewed as caching or pre-storing the inferences and recommendations that make sense at point in the space. Of course this assumes that the problem-solving space can be represented as a set of discrete states. As it is impossible to represent all the different action-paths that can be taken, a smaller set of paths, covering the most common approaches are used. These may be augmented with representations of paths that include common mistakes users make.

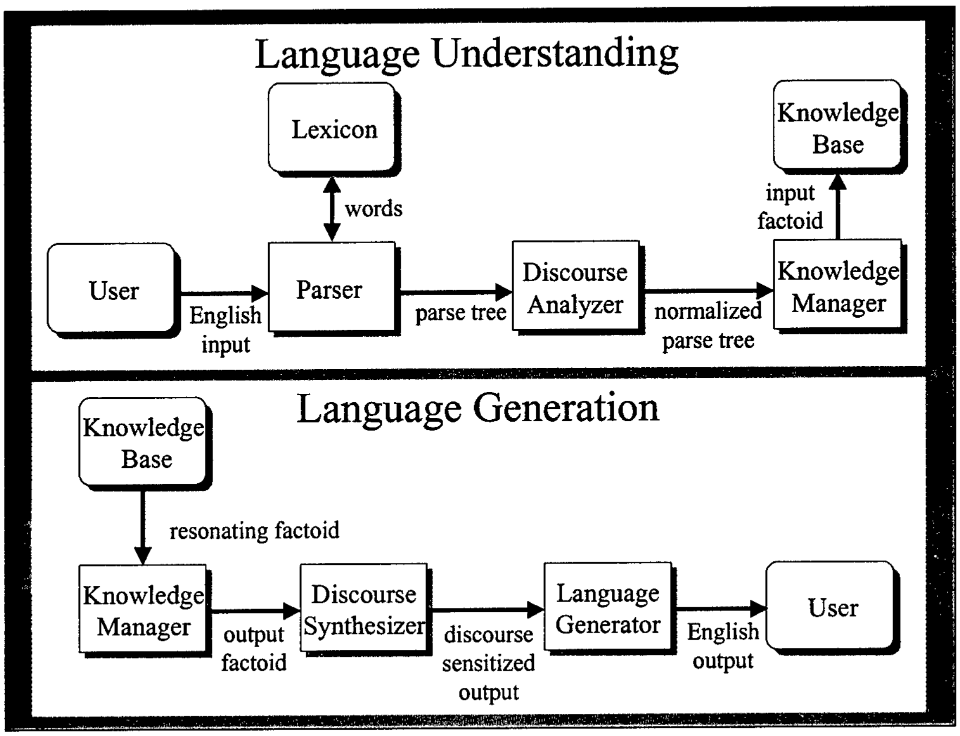
Mentor's effective problem space is even simpler than Sherlock's. The only additional annotations on problem-solving pathways are deductions that the subject matter expert would draw at that time. There are currently no annotations explaining why one action is better than another, or why another is inappropriate. Instead, Mentor simply recommends the next action. Finally, no paths to represent common student mistakes are part of Mentor's EPS.

There are more sophisticated artificial intelligence approaches to diagnostic troubleshooting. For example, Johan De Kleer's GDE (General Diagnostic Engine) can handle multiple faults. It represents the space of potential faults with assumptions about proper behavior of system components. Then a search is performed to find explanations that explain the actual observations in terms of a minimal number of faulted components. In contrast, the PROFILE diagnostic system by Towne and Munro simulates human troubleshooting behavior by taking into account the cost of tests, the likelihood of failure of particular components, and the cost of replacing components. De Kleer's approach can also be extended to take into account these measures, but each approaches the problem from a different vantage point. PROFILE was developed by studying actual troubleshooting experts. GDE was developed as an abstract general-purpose diagnostic engine. GDE requires a reasoning mechanism called an Assumption-based Truth Maintenance System (ATMS) to represent inferences that are true in different belief contexts (assumptions about which combinations of components are faulted). An information-theoretic approach can be used to select probes (tests or observations) that are optimal with respect to the total number of tests required. Further refinements can add in the measures that PROFILE uses that human experts consider (e.g., is it easier to just swap this component to see if it fixes the problem as it commonlyfails on this card?).

As mentioned earlier, the effective problem space is a simulation of problem-solving capabilities. It is easy to implement and to author but does not provide the robustness or depth of rule-based or model-based approaches to problem solving. Instead, each path in the effective problem space corresponds to one line of problem solving that eventually leads to a single diagnosis with optional actions after that. While there are multiple alternative paths and diagnoses, the strategy pursues one path at each decision point and performs no backtracking or other strategies. This is a limitation.

While we are capable of developing these more sophisticated AI and expert system approaches, the decision was made to focus proportionally more funds and effort on developing the dialog system and authoring capabilities.

We will briefly describe key dialog systems components and then give examples of capabilities and current limitations.



THE PARSER

In order to determine *who did what to whom*, the dialog system relies on its symbolic parser. This parser is capable of handling a broad range of clausal English constructions. For example:

a. SSG Bozo gave an insult to PFC Aiken.
b. He insulted PFC Aiken.
c. PFC Aiken was insulted by SSG Bozo.
d. It was PFC Aiken who was insulted by SSG Bozo.
1. What did SSG Bozo harass PFC Aiken with?

In *(a)* the semantic relations of the sentence are canonical. SSG Bozo is the *who* (the semantic "actor" or "agent"), an *insult* is the *what* (the semantic "patient" or "theme"), and PFC Aiken is the *whom* (the abstract semantic "goal", a polymorphic semantic role often realized as either a "beneficiary" or an "experiencer"). In *(b)* there is no overt *what* (the *what* has been incorporated into the verb as in "Sgt Bozo insulted an insult to Pfc Aiken"). In *(c)* the *whom* precedes the *who* (a "passive" construction). In *(d)* a passive construction is embedded within a "cleft" sentence, *It was X*. Although in *(b)* it is the task of the dialogue manager to identify who *He* is, in *(d)* it is the task of the parser to resolve *who* to *Pfc Aiken*. *(e)*, a question, illustrates that the order of the principal semantic elements can be permuted, here to *what-who-whom*. There are many more permutations of these fundamental elements and the many "oblique" elements of sentences (e.g., *when, why, how, where, if, because*). The parser must disentangle all of these before further processing is possible.

The parser recursively builds upon these clausal capabilities to perform metaclausal analysis, handling such constructs as subordinate/relative and coordinate clauses. Symbolic parsers can be divided into bottom-up and top-down types. Context-free bottom-up parsers are more efficient than context-sensitive top-down parsers if, and only if; 1) all interpretations of a sentence are of interest (e.g., *Flying airplanes can be dangerous*), and 2) there are no homonyms or polysemes in the sentence.

It is normally desirable, however, for a parser to be context-sensitive, so as to return only the most relevant interpretation (e.g., only *The flying of (or on) airplanes can be dangerous,* not *Airplanes which are flying can be dangerous*). Relatively uninflected languages like English are also rich in homonyms (e.g., *return* (v) vs. *return* (n)) and polysemes (e.g., *rich* (fatty, caloric) vs. *rich* (affluent) vs. *rich* (abundant)). Because conditions (1) and (2) are rarely encountered in English natural language processing tasks, top-down symbolic parsers are much more efficient in practice.

Top-down symbolic parsers can, in turn, be divided into two broad types: definite clause grammar (DCG) parsers and augmented transition network (ATN) parsers. Simple definite clause grammars consist of simple rules and are easy to author for small domains. Simple ATN grammars are more code-like and are harder to write, but as domains become large and grammars approach wide coverage, DCG grammars become less efficient and often harder to maintain than ATNs.

In light of the preceding considerations, DAS has adopted a Generalized Transition Network parser [Loritz 1993], which is an ATN parser extended by several DCG-like enhancements including shift-reduce mechanisms [Sato 1988], "fitted" bottom-up parsing [Jensen et al., 1993], deterministic "look-ahead" [Marcus 1980], a finite-state morphological transducer [Koskenniemi 1983] which is "cascaded" [Woods, 1980], and a gap-threaded hold list [cf. Alshawi 1992].

The parser currently parses with better than 95% syntactic accuracy on a wide range of constructions including English statements, questions, and imperatives, including passives clauses, inchoative and inceptive clauses, essive clauses, existential clauses, relative clauses, participial clauses and phrases, subordinate clauses, cleft sentences, pseudo-cleft sentences, and various gapped constructions. The grammar therefore meets the definition of a wide-coverage grammar. Parser error, when it does occur, is usually attributable to deficiencies in the lexicon.

THE LEXICON

An adequate lexicon is a major problem for all parsing systems. Large, pre-constructed online dictionaries can be adapted for parser use, but in this case there are at least three reasons why bigger, by itself, is not better. First, the lexicon ultimately needs to be coordinated with the ontology, and the larger the lexicon, the more difficult this task becomes. Second, as noted above, homonymy and polysemy seriously degrade a parser's performance. For this reason it is undesirable to have a large lexicon which is deeply coded with many out-of-domain word senses and sub-senses. Third, technical training almost always entails the learning of new technical terms and specialized jargon. As a result, no pre-constructed lexicon, no matter how large, is likely to have the most important words for the (instructional) task at hand.

Instead, the DAS parser operates with a core lexicon of the 4,000 most frequent English words and a main supplementary lexicon of the next-most frequent 16,000 English words [Carroll et al. 1971]. These combine to cover, in appreciable depth, some 30,000 word senses. Additional, domain-specific supplemental lexicons can also be called if an input word is not found in these master lexicons. The lexicon is a neural net in design, but only a semantic net in implementation -- there are no weights yet anywhere on the network nodes.

The upshot of all these design considerations is that the DAS lexicon must be authorable. The DAS lexicon was designed for authorability from the ground up. To simplify lexicography, its lexicons were organized as semantic networks: taxonomies into which new terms can be added and from which default lexical features can be automatically inherited [Loritz 1993].

THE REPRESENTATIONAL HIERARCHY

The DAS knowledge-representation schema is structured as a three-level hierarchy. The base of the representational hierarchy is made up of *alpha-objects*. We have encountered these workhorses of knowledge representation before, under the colloquialism of "factoid". By whatever name, these objects represent simple propositional relationships pertaining among the primitive concepts in the universe of discourse, the equivalent of simple declarative clauses (save that they are composed of language-independent *ideas*, rather than words). Factoids corresponding to "it is raining today" or "Arthur canceled the Sunday-school picnic" typify the KR propositions found at the alpha level. Unlike the building blocks used in some other knowledge representation systems, however, alpha-objects are well suited to modeling propositions about individuals (extensional knowledge) as well as about classes (intensional knowledge), making it possible to override inherited properties and thereby support non-monotonic modes of reasoning. A more extensive discussion of alpha-level factoids may be found in [DeSmedt 1995].

A middle layer of *beta-objects* represents simple, binary relationships between alpha-factoids. One common use for beta-linkage is the "because" conjunction: given the two factoids corresponding to "it is raining today" and "Arthur canceled the Sunday-school picnic," the proposition "Arthur canceled the Sunday-school picnic because it is raining today" is a beta-object.

The apex of the KR pyramid is occupied by *gamma-objects*. They generalize the beta-link structure by accommodating an arbitrary number of alpha-, beta-, or (recursively) gamma-objects in a wide range of relationships -- deterministic and probabilistic logical and relational operators, implication, universal and existential quantification, etc. Since they can quantify over operands which are themselves gamma-relationships as readily as over first-order variables, the expressive power of gamma-objects is equivalent to that of second-order predicate calculus.

It is at the gamma level that the DAS knowledge representation schema attains the ability to capture common-sense propositions like "no one can be in two places at once." More precisely: "for all x, y, z, where (x is a person) and (y is a location) and (z is a location), if (x is at y) and (y is not equal to z) and (y does not contain z) and (z does not contain y), then (x is not at z)" That is, this deceptively simple commonsense rule requires eight atomic propositions, five relational operators (six, counting the "not"), an implication, and three universal quantifications.

TAXONOMIC KNOWLEDGE

Perhaps surprisingly, one aspect of common knowledge that is *not* represented in the CKB — or in any other of the DAS-managed knowledge bases — is taxonomy: the relationship of individuals to classes ("Bob is a golden retriever") and subclasses to their superordinates ("A golden retriever is a dog," "a dog is a mammal"). Such taxonomic relations, dubbed *is-a* links for obvious reasons, are frequently "the most common type of link" in traditional knowledge representation systems [Evett, Hendler, & Spector 1990]. The difficulty is that this architecture compels those KRSs adopting it to commit a goodly proportion of their resources to rediscovering time and again that, e.g., a dog *is-a* mammal.

DAS, on the other hand, employs an explicit *type hierarchy* to represent taxonomic relations. This involves encoding all the information normally contained in *is-a* links into the literal values for the concepts instead. The result is that the class/member relationship between the concepts for "dog" and "mammal" ceases to be an explicit proposition in the knowledge base, and becomes an implicit property of the constant assigned to "dog" in relation to the one denoting "mammal". This results in considerable representational economy, even as it provides universal and existential quantification — along with many of the mechanisms of "inheritance reasoning" — to all intents and purposes *for free.*

SEMANTIC DISAMBIGUATION

The parser will disambiguate nouns versus verbs which are the same word – bat as a noun, and bat as a verb. But it will not [usually] disambiguate "bat" as two different nouns. The parser will not 'know' whether "He saw the bat" is meant as a baseball bat or a winged bat. It leaves this task to be resolved by the present (restrictive) domain.

Using its ontolexicon, MetaLang can filter out parses from EngPars that do not satisfy semantic restrictions on verb cases. Thus while the "bat" is ambiguous in "He saw the bat", the "The bat flapped its wings." is not ambiguous. In such a case, EngPars generates both possible parses (either kind of bat) but MetaLang enforces semantic restrictions as only living creatures typically flap wings so the animal-bat interpretation is the only one that makes sense.

VARIABILITY IN UNDERSTANDING

The Dialog system "understands" a range of answers that are semantically equivalent regardless of different wording and syntax, although there are limits to what is handled. It can recognize verbs or nouns as instances of more general categories. Thus it can recognize that a gun is a weapon and answer a question like "Do you have a weapon?" with "Yes, I have a gun." or "Yes, I have a knife."

The Dialog system can resolve the following kinds of variability,

Most **anaphoric** reference including (a) *pronomial reference* ("It beeped") and (b) *sentential reference* ("Yes." "It did.""It is.") constructions that need to be resolved in terms of antecedent references, principally: personal pronouns (*he, she, it*), propositional anaphora (*I know that*), one-anaphora (*One of our aircraft is missing*), deictics (*this, that, here, there, now, then, you, me*), and definite descriptors (*What did the doctor say then?* referring to a previously mentioned physician).

synonyms (computer=workstation=machine), provided these have been specified in the ontolexicon

hierarchical reference, both for (a) *nouns* (given that "Schmidt has a knife", handles "Does Schmidt have a weapon?") and (b) *verbs* (equating "my machine can talk to other machines" to "my machine can communicate to other machines")

ellipsis, sentence fragments often occurring as requests for an elaboration of the preceding statement (*How?, Why?, With whom?*).

CURRENT PARSER AND SEMANTIC LIMITATIONS

Inevitably there are limitations in what can be handled. Some are due to limitations in the parser (EngPars), some are due to limitations in semantics (MetaLang) and some are due to the interactions between the two.

Concept names must be unique in the ontolexicon for a particular category of speech otherwise confusion between concepts can cause problems when the same word could have multiple word senses. For example, "orange" appears as a noun under "citrus" but does not also appear as a color. It can be added as a noun under color, too. But it is not clear that the parser can distinguish the two senses (e.g., the sentence "The color of your shirt is orange." is proofed to "The shirt's color is an orange."). [note: it does take, "You have an orange shirt"]. Similarly, "tank" appears under a category for armored transportation but not under containers, so the military sense of tank precludes including the water tank sense of tank, at least with the same name and with no possibility of confusion. It could be included in the ontolexicon as the phrase "water tank".

The same word can be used for different concepts if the word is used in different categories of speech. Thus, you can enter *orange* as a noun and as an adjective, or *tear* as a verb (to rip) or as a noun (as from crying).

Prepositional Phrase Attachment. Although the parser can handle individual prepositional phrases, either the parser or the internal handoff to semantics component has problems with multiple prepositional phrases. This is a classic NLP problem. Without considerable context, prepositional phrases are highly ambiguous, as in the sentence *I saw a man on a hill with a telescope.* – do *I, the man,* or *the hill* have *the telescope*?

Examples of sentences not handled:

You are in a tent in the desert in North Africa.

You live in a house by a stream in the forest.

Workarounds:

You are in your tent.

Your tent is in the Sahara desert.

The Sahara desert is in North Africa.

With the second set of 'workaround' factoids, the DAS should produce correct answers to questions such as "Are you in the Sahara Desert"? "Is your tent in North Africa?"

Cascaded adjectives or adverbs. There are similar problems with multiple adjectives, adverbs, or relational clauses.

Examples of sentences not handled:

You have a heavy steel Japanese sword.

The rat that was on your hat that was on the door ran away.

Workarounds:

You have a heavy sword.

The heavy sword is Japanese.

The rat was on your hat.

Your hat was on the door.

Examples of variants handled in the Mentor Sun workstation problem space. Checkmarks indicate the variants that were handled by the dialog system and "X" marks those variants that were not handled in the current implementation.

Introduction: Hello, Bill. √

Question 1: Did your keyboard beep?
Variants tested:
1. *Yes.* √
2. Yes, *it* did. √
3. *It* did. √
4. The keyboard beeped. √
5. My keyboard beeped. √
6. The keyboard of my workstation beeped. X
7. It beeped. √

Question 2: Did your system start to boot?
Variants tested:
1. *Yes.* √
2. *It* started. X
3. Yes, *it* started booting. √
4. *Affirmative.* √
5. My *workstation* started booting. X
6. This system started to boot. √
7. The *computer* started booting. X
8. This *system* started booting. X
9. *The boot process* started. X
10. *It* started to boot. √

Question 3: Did text appear on your display?
Variants tested:
1. *Yes.* √
2. Yes, *it* did. √
3. Text appeared on the *monitor.* X
4. Text appeared on my monitor. √
[here an issue of *my* monitor, versus *any* monitor referred to as 'the']
5. Some text appeared on my display. √
6. I saw some text there. X
7. A few lines of text appeared. X
8. Some characters of text appeared on my display. X
9. My display showed some text. X
10. Some text is there. X
11. Text did appear on my display. √

Question 4: Does the memory initialization procedure complete?
Variants tested:
1. *No.* √
2. *It* did not complete. √
3. *It* failed to finish. X
4. The memory initialization procedure failed to complete. X
5. The memory initialization procedure fails to complete. X
6. *It* didn't. X
7. *It* did not terminate. X
8. The initialization procedure did not complete. √

Mentor Conclusion: There may be a problem in the SIMM banks: a missing, loose, or bad SIMM.

REASONING

Special-purpose reasoning mechanisms are used for efficient reasoning in MetaLang. Inheritance reasoning supports quick determination of hierarchical references by traveling up ontologies from individuals or concepts to more general concepts.

MetaLang uses an underlying clausal form and has reasoning capabilities equivalent to First Order Predicate Calculus. Unfortunately these are not fully available to knowledge base authors due to restrictions on parsing sentences with negation and existentials (see below) and converting the parsed sentences into a logical form for input to MetaLang. Thus, from the viewpoint of a MENTOR author, only modus ponens and inheritance is currently available.

Factual Reasoning

The primary kind of built-in reasoning in MetaLang is inheritance. For example, if Fred is a bat, and a bat is a mammal, mammals are warm-blooded, and therefore Fred is warm-blooded. Antecedent reasoning (modus ponens) is also supported. For example, given the factoids below:

Van Gogh is an artist.	artist(van_gogh).
All poor people eat sandwiches.	if poor(x) then eats(x,sandwiches).
All artists are poor.	if artist(x) then poor(x).

MetaLang can deduce poor(van_gogh) and eats(van_gogh,sandwiches) where van_gogh stands for the individual whose name is "Van Gogh".

CURRENT REASONING LIMITATIONS

Currently, the dialog system does not handle negation and thus cannot handle contrapositive reasoning. For example it does not have the reasoning required to deduce that Bill Gates is not an artist from the following factoids and rules.

Bill Gates does not eat sandwiches.	not(eats(bill_gates,sandwiches)).
All poor people eat sandwiches.	if poor(x) then eats(x,sandwiches).
All artists are poor.	if artist(x) then poor(x).

In an earlier build, Mentor made closed world assumptions. If it did not have a fact in its knowledge base, then it would answer in the negative. For example, if it did *not* have the fact, The Pope is Catholic, then Mentor would answer the question, Is the Pope Catholic? with "No, the Pope is not Catholic." The handling of such questions has been changed so that Mentor no longer makes a closed world assumption. Now, it would answer, "I do not know if the Pope is Catholic."

Much of what is not handled is due to its restricted set of common-sense rules about word senses and restrictions on inference capabilities. For example, if *x* kills *y* then we (humans) know *y* is dead, but we must add a rule for MENTOR to know this, too. In addition to the need for common-sense reasoning rules (a huge project as shown by CycCorp's work on Cyc) there is no inference other than inheritance and modus ponens available to a MENTOR author, even though the underlying knowledge representation system MetaLang can handle all the inferences of any first order predicate calculus system.

EXAMPLES OF WHAT IS NOT HANDLED

No reasoning system can handle all kinds of reasoning, including humans. Following is a summary of some of the reasoning that is not currently handled by the Mentor system.

Negation. Currently one cannot author rules or facts that express that something is not the case or never can be. There is some reasoning of this type already encoded as factoids in the Mentor system. For example, there is a built-in semantic transform (procedure) that allows Mentor to deduce that if you hate something you do not like it, and if you are sane you are not crazy:

But rules such as

All poor people are not rich.

cannot be entered into Common Knowledge.

Existential sentences. Sentences that start with "There is.." are not proofed accurately.

There is a pink Christmas tree.
There is a flea on that dog.

Beliefs about propositions. Beliefs about propositions cannot be asserted in sentences. For example, the following cannot be handled

This sentence is false.
John believes everything.
Mary believes nothing John tells her.

COMMON-SENSE INFORMATION THAT MUST BE AUTHORED

The dialog system does not have a large common-sense knowledge base, but it does have an authoring mechanism for this. Until a larger common sense knowledge base is authored, the Dialog system will fail to understand some items that users might expect. Some examples follow: Please note that there is no working system (anywhere) that currently contains all of these types of reasoning and knowledge that human beings use so effortlessly.

Rules about words and what they mean. The meanings of words and their relationships must be explicitly provided. For example, one can author these facts, "You are dead" and "You are alive" and then the character will answer affirmatively to both questions. It has no knowledge that one excludes the other without specific authoring about that fact. There is no information that some states preclude others or that some adjectives imply others

Rules about roles and what they mean. Similarly, the system has no built-in knowledge of social or work roles. For example, it has no knowledge that *x* manages *y* implies that *y* works for *x*, that all wives are women and all husbands are men, that *x* teaches *y* implies *y* learns from *x*, and so on. All such information can be provided with other constraints, but the lack of negation makes it difficult for one term to rule out another.

Temporal reasoning. Although the system has a representation of time, it is fairly primitive. It cannot reason about temporal intervals or whether one interval meets or overlaps another. (You were born in the afternoon. Were you born in the morning?)

Spatial reasoning. The system cannot perform any diagrammatic or map-based reasoning. It needs explicit knowledge that relationships such as "taller" and "larger" are transitive. It does not know how North, South, East, and West are related.

Reasoning about emotions and plans of people. It does not know what motivates people and why they might act the way they do.

Reasoning about natural kinds. The built-in knowledge about natural kinds is limited to what is in Common Knowledge. Particular individuals can be exceptions to the rules (e.g., "Bowser is a dog with three legs.").

Reasoning based on world knowledge. Given "You were born in the afternoon." it cannot answer the question, "Where were you that same morning?" Knowledge about stereotypical situations (eating in a restaurant, childbirth, using an elevator, marriage, death, etc.) must be supplied.

RELATED WORK

In this section we discuss related work. First we consider the historical antecedent to the Mentor conversational system, Herr Kommissar. Next we discuss intelligent tutoring systems with natural language interfaces and other conversational natural language systems. Finally we discuss the problem of common-sense reasoning and how Cyc and other systems have approached it.

HERR KOMMISSAR

Herr Kommissar (the Police Commissioner) is an educational game to teach German [DeSmedt, 95]. A student plays the role of a detective visiting a police station and attempts to solve a murder by interrogating several suspects. The student must interrogate them in German and Herr Kommissar checks their input for grammaticality. It provides corrections if slips are made and it can determine what was most likely intended. It also keeps track of the mistakes and kinds of constructions used to develop a model of the user's grammatical coverage and capability.

The Herr Kommissar domain is highly constricted by the game setup, but these restrictions do not intrude upon the game play in the same way as they do when the same or similar restrictions turn up in Mentor and restrict authoring. For Herr Kommissar it makes sense that the NLU need only focus on answering questions, as the different conversational characters play the roles of different subjects being interrogated. We do not expect a police commissioner to be volunteering much information, instructing the subjects of interrogation, or trying to handle *their* questions or requests. Instead, we expect the commissioner to ask questions of the suspects and try to detect contradictions in their statements.

Authoring new cases is similarly simplified by the game context. To define a new case to solve we need only define new characters with differing personalities (to make the game interesting), and each with a different set of beliefs about what happened during the period of the murder and about the other characters. We do not expect the characters to have broad general knowledge or to perform sophisticated reasoning. Rather we expect them to be obstructive (if they are guilty or do not like the police in general), guarded, and depending on the character, possibly to be of average or below intelligence.

INTELLIGENT TUTORING SYSTEMS WITH NATURAL LANGUAGE INTERFACES

Some of the early work done in intelligent tutoring systems used natural language interfaces. Mentor is not an intelligent tutoring system (ITS) but since it is a performance support system using an NL interface it is worth comparing NL work in ITS.

The SOPHIE [Brown, Burton, and de Kleer, 82] system allowed students to practice troubleshooting a power supply. The NL interface used a semantic grammar [Allen, 95] to handle questions in a highly restricted subject area: troubleshooting for a particular Heathkit power supply. Students could ask questions about voltage and current, request part replacements, etc. Students could also volunteer a hypothesis as to what the circuit fault was. A SPICE circuit simulator was used to provide the answer to questions (it would simulate the faulted system and all the voltages and currents that could be tested). SOPHIE could also apply basic laws of electronics (e.g., summing voltages around a loop or currents into a terminal) to determine whether a measurement could already be predicted and was thus unnecessary. It could also determine if particular faults the student hypothesized were consistent or not with the measurements made so far. But the key aspect concerning us here is its NL interface. By using a highly restrictive domain SOPHIE avoided the need for common-sense reasoning, authoring new terms, and authoring new facts. Its semantic grammar interface was also highly tolerant of ungrammatically (e.g., sentences like, "What is the volts measured at T23?").

Carbonell's SCHOLAR [Carbonell, 70] system tutored students in geography. It used a Socratic Question and Answer instructional strategy, an instructional strategy that is hard to realize and rarely used in non-NL ITS systems. The WHY [Collins, 75] system taught meterology using an NL interface.

CYC AND OTHER COMMON REASONING SYSTEMS

Cyc [Lenat, 95] is a very large ontology and knowledge-based reasoning system. It is intended to be a substrate for building more robust knowledge-based systems that are less brittle than typical expert systems due to the greater depth and breadth of their knowledge. It is much more than a large taxonomy of concepts, which indeed would not fit the formal definition of an ontology. Typically, each concept in Cyc has about 10 axioms (rules) that constrain its meaning. So much of what would need to be added to Mentor to provide a common-sense infrastructure to support specialized subject matters is present in Cyc in the form of these 1000s of rules

Cyc is a huge effort, involving millions of dollars of research spent over a period of 10+ years. It was first developed at MCC under investor funds and is now being developed at CycCorp. Such a huge effort dwarfs the scope or funding of this SBIR project. It does provide a useful comparison point, though bearing in mind that orders of magnitude more effort and funding have gone into Cyc.

Although Cyc is being used to develop a natural language system by Cycorp, it is significant that there is no such system yet, after 10 years of research, and prior work on NLP and Cyc at places such as MCC. Cyc has simple text generation capabilities using templates and a small NL-interface was built to facilitate authoring in Cyc. In practice, the NL-interface produces dozens of parses for even small sentences and the Cyc knowledge engineer must still determine the proper Cyc term to use for an English word.

For example, a simple set of assertions and questions such as the following:

> Pengi is a penguin.
>
> Can Pengi fly? (expected answer: Yes.)
>
> Pengi has a broken wing.
>
> Can Pengi fly? (expected answer: No.)
>
> Pengi is at the airport.
>
> Can Pengi fly to Boston? (expected answer: Yes)

quickly becomes difficult to encode into Cyc primarily as the author must specific exactly what kind of flying is referred to in each instance. One kind of flying is that performed by winged animals and another kind that performed by aircraft. The first kind is called **Flying-FlappingWings** and the second is represented by the Skolem function[1] **(TransportViaFn AirTransportationDevice)**. The key point here is that although Cyc has made significant progress in representing common-sense knowledge its authoring is non-trivial, far more difficult than Mentor's. Mentor's authoring is designed for non-programmers whereas at present Cyc must be authored by knowledge engineers with an understanding of ontologies, the subject matter to represent, and the Cyc system itself.

CONVERSATIONAL NATURAL LANGUAGE SYSTEMS

Finally, let us consider other natural language systems outside of ITS. First we consider systems that can be called chat agents or search agents, as they can provide a simple chat or document search function without a real understanding of text. Second, we consider systems that attempt to understand text by interpreting it into an internal semantic representation.

Chat and search agents, with no internal semantic representations

Chat and search agents provide limited but useful functionality without understanding the text they deal with. Chat agents search for patterns in text and then construct replies from reply patterns stored with the text. Eliza is the most famous example. More sophisticated versions, such as Julia, use refinements such as increasing rule activations for rules consistent with current topics, to provide an increasingly coherent conversation. Such systems tend to be limited to toys or simple guides, e.g., Julia acts as guide to the LambdaMOO MUD, and Extempo's characters were used as web guides to discuss products on corporate sites and to encourage return visits.

Search agents can use information retrieval (IR) techniques to throw out noise words in queries and then match the proportion of relatively uncommon words in documents to those in queries to find likely matches. Most web search agents use such techniques.

LSA (latent semantic indexing) is a statistics-based approach to determine the similarity of documents. In both cases the words could be uninterpreted symbols, except for the use of a few tricks such as stemming (dropping suffixes from words to get to a more canonical base form).

Systems that parse to a semantic representation

NL interfaces to databases can be built with simple semantic grammars. LADDER [Hendrix, 78] provided a semantic grammar for querying about the location and status of naval assets on the oceans. With a semantic grammar the terms of the grammar directly correspond to meaningful terms in the domain (e.g., "ship", "capital ship", "cruiser", etc. for the naval domain).

Microsoft now supplies an English-to-SQL interface that uses a similar semantic grammar approach. The user defines a grammar of English words that defines a mapping of terms to database tables, fields, and attributes and English queries to SQL queries. Once a query has been mapped to SQL the SQL query is carried out and the results returned. In this case SQL is the underlying semantics.

---

[1] A skolem function maps from an individual to some associated entity, e.g., from each person to their nose. In this case it maps from an air transportation device to the transport function performed by that device.

CONCLUSIONS

In this section we discuss conclusions from the research on the Mentor and Conversational Agents project. We briefly discuss the benefits and limitations of the approach to NL-based performance support, and then consider how to build on these findings in future projects.

BENEFITS OF MENTOR'S NL APPROACH TO PERFORMANCE SUPPORT

A natural language interface is engaging

Mentor is engaging because you can enter natural dialog and it demonstrates a level of understanding. Conversational characters can be entertaining even when they make dialog-based errors. Part of the rationale that Extempo used to sell its web guide characters was that they would increase return visits because users enjoy interacting with the characters. Their informal studies verified this intuition. Similarly, Mentor shows some of the appeal that conversational characters can have. It can be enjoyable to interact with Mentor just to find out its limits, and what it does and does not know. The key point is that a natural language interface may be more motivating for users then a standard point-and-click interface, increasing a user's desire to use a performance support system.

A natural language interface allows for role-playing in instruction

Two different kinds of character roles were developed for the SUN will-not-boot troubleshooting domain: an instructor role and a trainee role. A Mentor agent, called LT Acer, takes the role of instructor. LT Acer is highly directive, taking the lead in asking questions and recommending actions for the trainee. A conversational character, called SGT Klutz, plays the role of a trainee. Users can practice troubleshooting skills by asking SGT Klutz to perform various tests to uncover the problem.

Although the role-playing is imperfect, it points to new capabilities for training as mentors and instructors and through role-playing characters. For foreign-language instruction, the conversational characters could model different roles that could be encountered in a village (vendor, bank teller, mailman, policeman, restaurant owner, waiter, etc.). For prisoner interrogation different kinds of prisoners could be modeled (cooperative, uncooperative, deliberately deceptive, etc.).

A natural language interface for authoring simplifies entering facts

Entering facts by typing in English sentences is much easier for non-programmers than using formal languages, such as First-Order Predicate Calculus, or stylized restricted artificial languages. The Mentor system allows this kind of authoring, albeit restricted by its parser and reasoning capabilities. When all terms are defined, and a sentence is parsed and correctly understood, authoring in Mentor is much easier than in other knowledge-based systems or ontology-based systems. The Mentor approach obviously has much more potential to open up authoring directly to subject matter experts and other non-programmers.

Extending an ontology by adding to a tree of terms is relatively straightforward

Adding simple subcategories and lexical terms for them to Mentor is also quite straightforward. One need merely search for a more general class that is already defined and then add the subcategory as *a kind of* subclass of that category. Again this approach is easy for non-programmers, although the particular GUI used to implement this approach could obviously be improved. Ideally, the tree could be displayed graphically, with zooming, auto-scrolling and new nodes could be dragged under existing nodes.

ASPECTS THAT NEED IMPROVEMENT

Tolerance of variant input

The dialog system tends to reject from a third to a half of what an untrained user would enter in normal conversation. By reject, we mean either the input does not parse, is rejected as logically unacceptable, or is accepted but the proofed sentence is not a paraphrase of the input, but instead has a different meaning. These scope limitations are largely because Mentor has limited built-in deductions and therefore the onus is on the author to supply all the required information.

Limitations in common-sense reasoning

Providing a common-sense ontology with both concepts and axioms defining them is a prerequisite for authoring new domains. It is clear from the Dialog System's limited ability to understand variant input that common-sense reasoning is important just to understand normal English text, let alone real-world situations and their consequences, or the plans and actions of people. Furthermore, although this common-sense knowledge could be authored into the Dialog Agent system (provided its reasoning mechanisms were extended, as needed) it is a tremendous job far beyond the scope of non-programmers. The Cyc project gives an idea of how large this task is.

Limitations in authoring for a new subject area

An ontology is more than just a rich taxonomy. Without axioms (rules) to constrain and relate the meanings of the terms, reasoning is very restricted. Leaving this task to a non-programmer author is unrealistic, it must be provided beforehand.

FUTURE DIRECTIONS

Mentor points to promising new directions in performance support. Conversational characters can play multiple roles in different performance support applications. A conversational interface, when it works well, can be more engaging and expressive than a conventional interface. It is especially relevant when the target task involves language itself (e.g., translation and interrogation) or acting in a role-appropriate way in certain situations (e.g., detective or police work).

Such a system must be robust if the interface is to be helpful rather than get in the way of what the user wants to do. Robustness in natural language understanding requires common-sense reasoning. Common-sense reasoning, in turn, requires a large coherently organized ontology comprising both terms and axioms restricting the meanings of the terms. With the advent of systems like Cyc and the Standard Upper Ontology effort it should be practical to provide such an ontology in the near future. The result should be a better kind of performance support system, one that can handle natural language and multiple roles robustly. Such a system would also be better suited to reason about subject matter material based on facts authored in English, as all terms would be grounded in the common-sense ontology.

REFERENCES

[Allen, 95] Allen, J. Natural Language Understanding. Second Edition. The Benjamin/Cummings Publishing Company, Inc.

[Brown, Burton, and de Kleer, 82] Brown,J.S.; Burton, R.R.; and de Kleer, J. Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II, and III. In Intelligent Tutoring Systems, eds. Sleeman, D.; and Brown, J.S. Academic Press. 1982.

[Carbonell, 70] Carbonell, J.R. *Mixed-Initiative Man-Computer Instructional Dialogues.* Doctoral dissertation, MIT. Cambridge, Massachusetts.

[Collins, 75] Collins, A. Analysis and synthesis of tutorial dialogs. In Bower, G. (Ed.) The Psychology of Learning and Motivation (vol. IX). Academic Press, New York.

[DeSchmedt, 95] DeSchmedt, W. H. Herr Kommisssar*: An ICALL Conversation Simulator for Intermediate German.* In Intelligent Language Tutors: Theory Shaping Technology. Editors Holland, Kaplan and Sams. 1995. Lawrence Erlbaum Associates, Publishers. pp. 153-174.

[De Kleer and Williams, 87] De Kleer and B. C. Williams. Diagnosing Multiple Faults. Artificial Intelligence. Volume 32, Number 1. April 1987. pp. 97-130.

[Extempo]. www.extempo.com

[Hendrix, et al., 78] Hendrix, G., Sacerdoti, E., Sagalowicz, D., and Slocum, J. 1978. Developing a natural language interface to complex data. ACM Transactions on Database Systems 3, 2:105-147.

[Lenat, 95] Lenat, D. B. "Artificial Intelligence." Scientific American (September 1995).

[Lesgold, et al., 92] Lesgold, A., Lajoie, S. P., Bunzo, M., & Eggan, G. (1992). A coached practice environment for an electronics troubleshooting job. In J. Larkin, R. Chabey, & C. Cheftic (Eds), *Computer assisted instruction and intelligent tutoring systems: Establishing communication and collaboration* (pp. 201-238)). Hillsdale, NJ:Lawrence Earlbaum Associates.

[Loritz, 95] Loritz, D. *GPARS: A Suite of Grammar Assessment Systems.* In Intelligent Language Tutors: Theory Shaping Technology. Editors Holland, Kaplan and Sams. 1995. Lawrence Erlbaum Associates, Publishers. pp. 121-133.

[Murray, 90] Murray, W.R. A Blackboard-based Dynamic Instructional Planner. In Proceedings Eighth National Conference on Artificial Intelligence. pp. 434 - 441, 1990. AAAI Press / MIT Press.

[Holland, Kaplan, and Sams, 95] V. Melissa Holland (Editor), Jonathan D. Kaplan (Editor), Michelle R. Sams. Intelligent Language Tutors: Theory Shaping Technology.

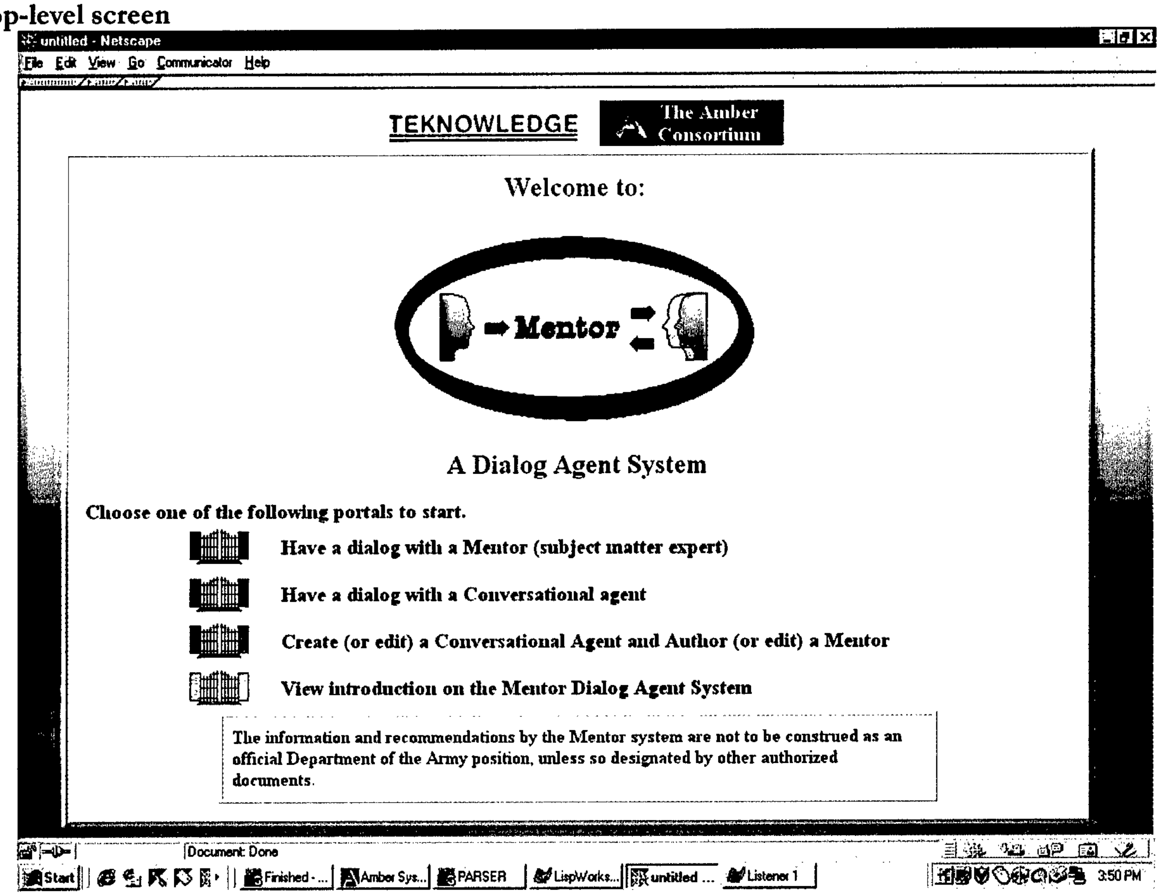
[Towne et al., 87] Towne, D.M.; Munro, A.; Pizzini, Q.A.; and Surmon, D.S. Simulation composition tools with integrated semantics. Abstracts of the Third International Conference on Artificial Intelligence and Education, p. 54. Learning Research and Development Center, University of Pittsburg, Pittsburgh, Pennsylvania. .

[Wenger, 87] Wenger, E. Artificial Intelligence and Tutoring Systems—Computational and Cognitive Approaches to the Communication of Knowledge. Morgan Kaufmann.

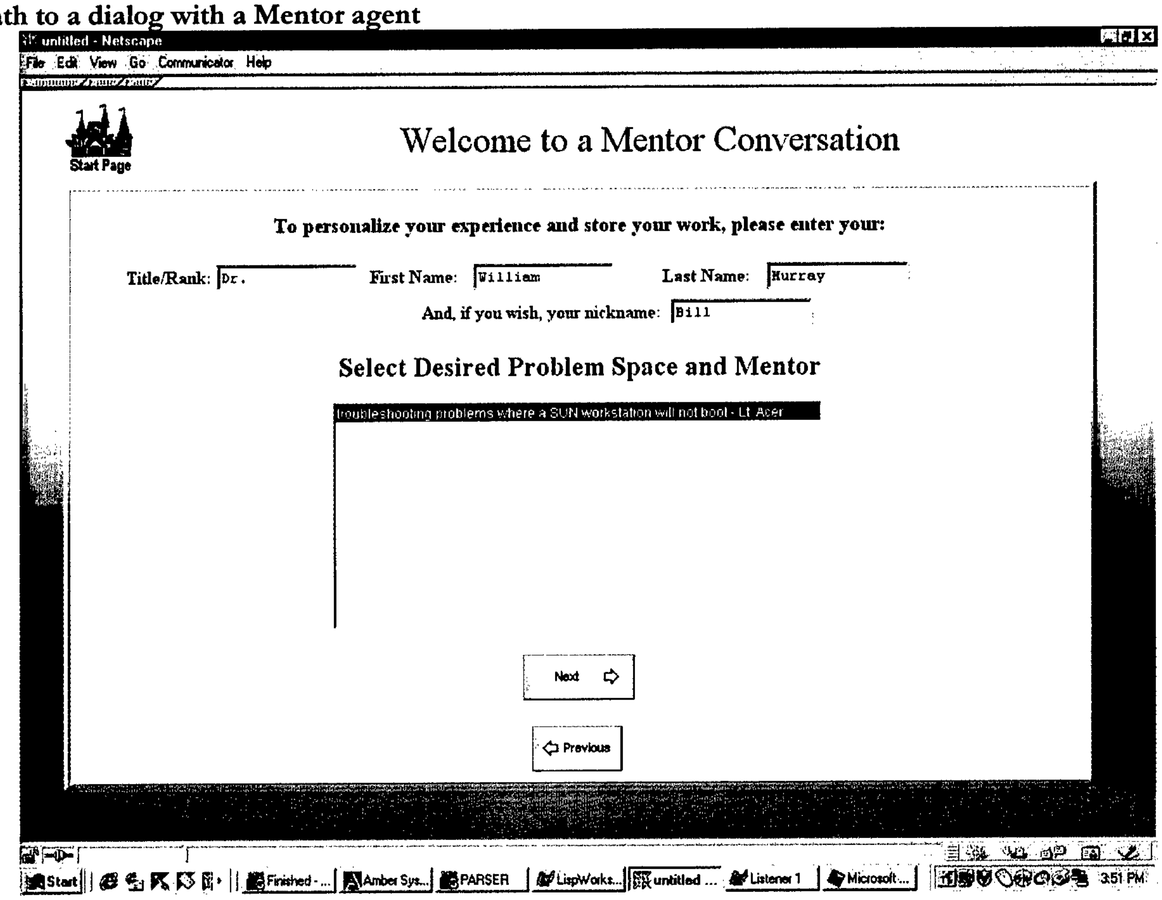
APPENDICES

MENTOR GUI PATHS TO AUTHORING AND CONVERSATION

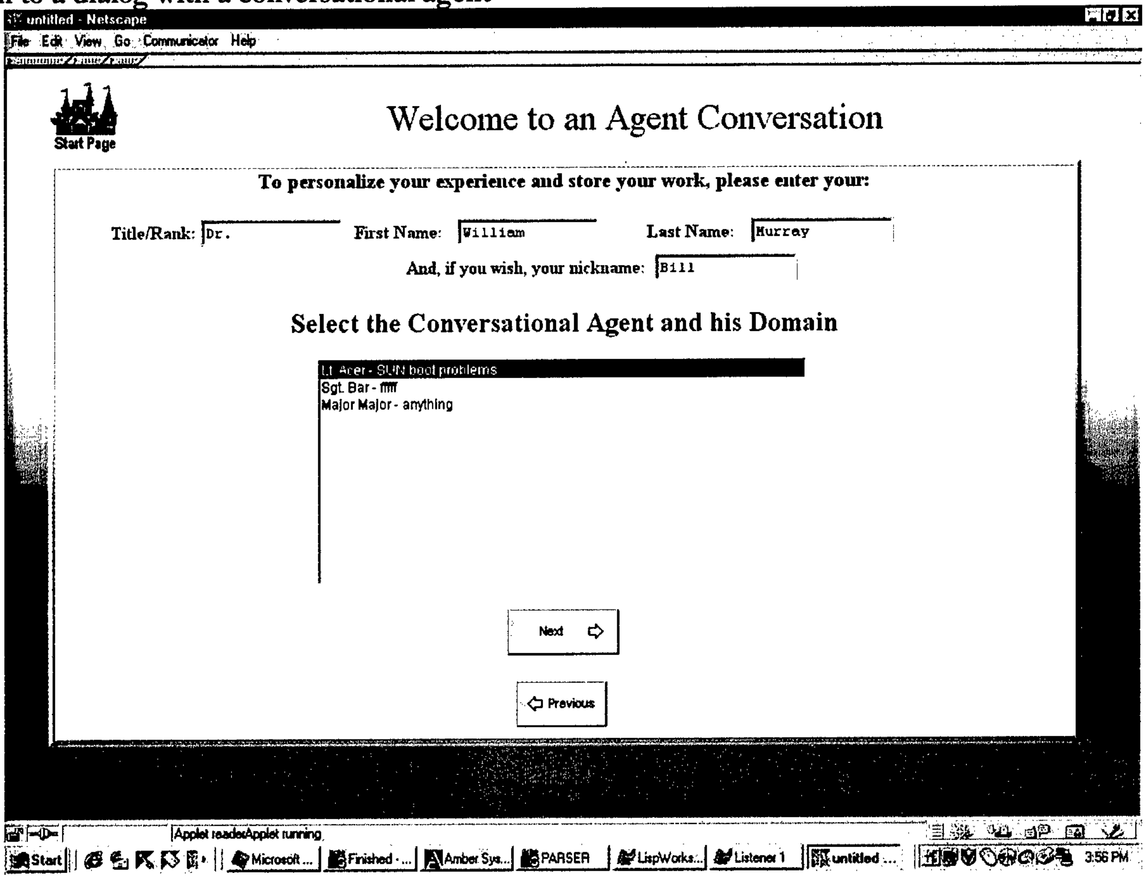
Top-level screen



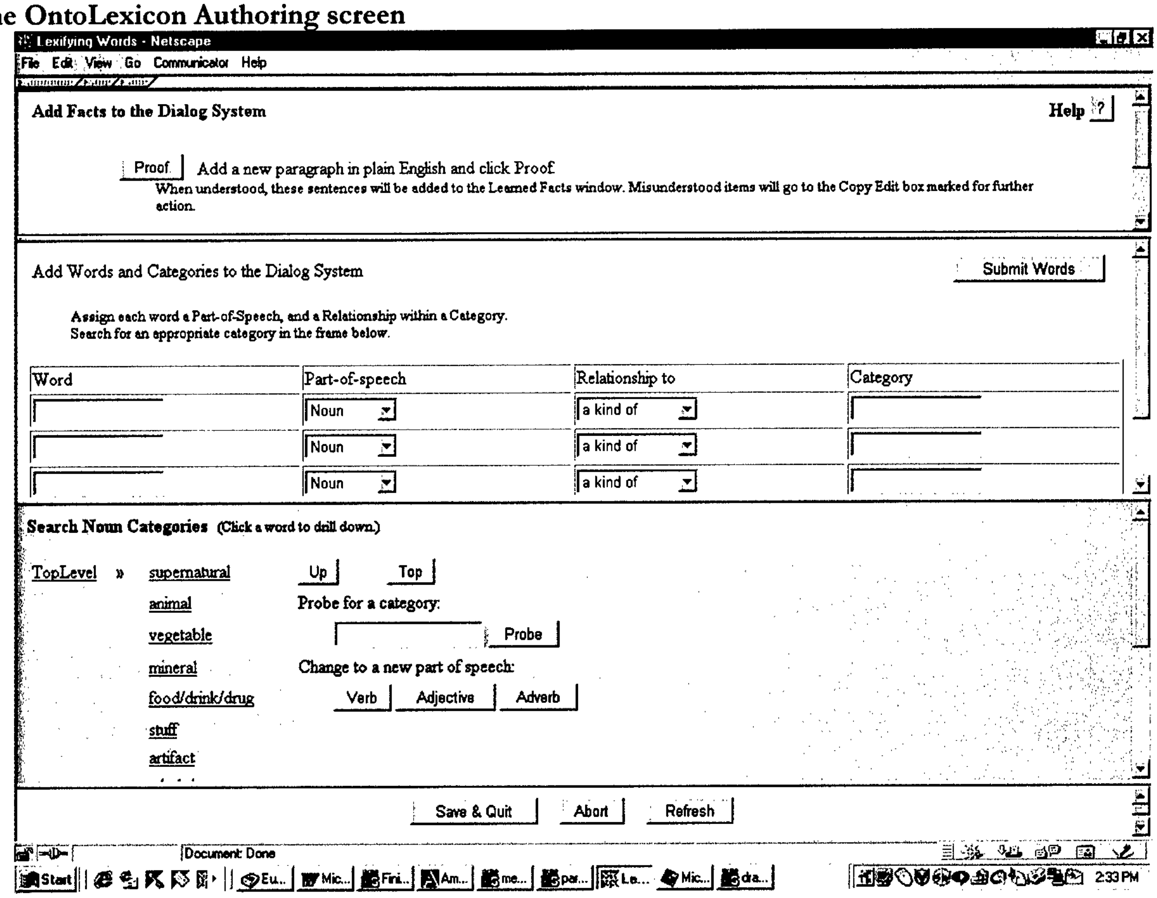
Path to a dialog with a Mentor agent



Path to a dialog with a conversational agent

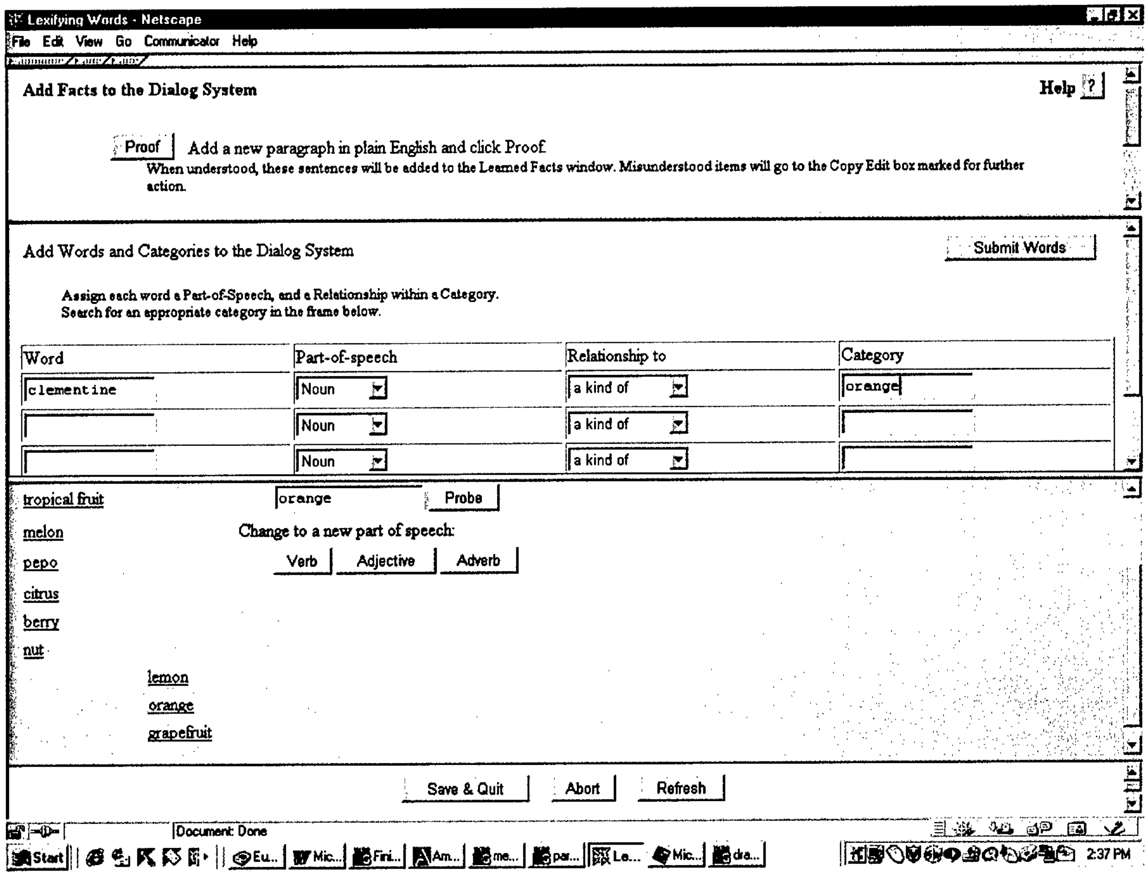


The OntoLexicon Authoring screen



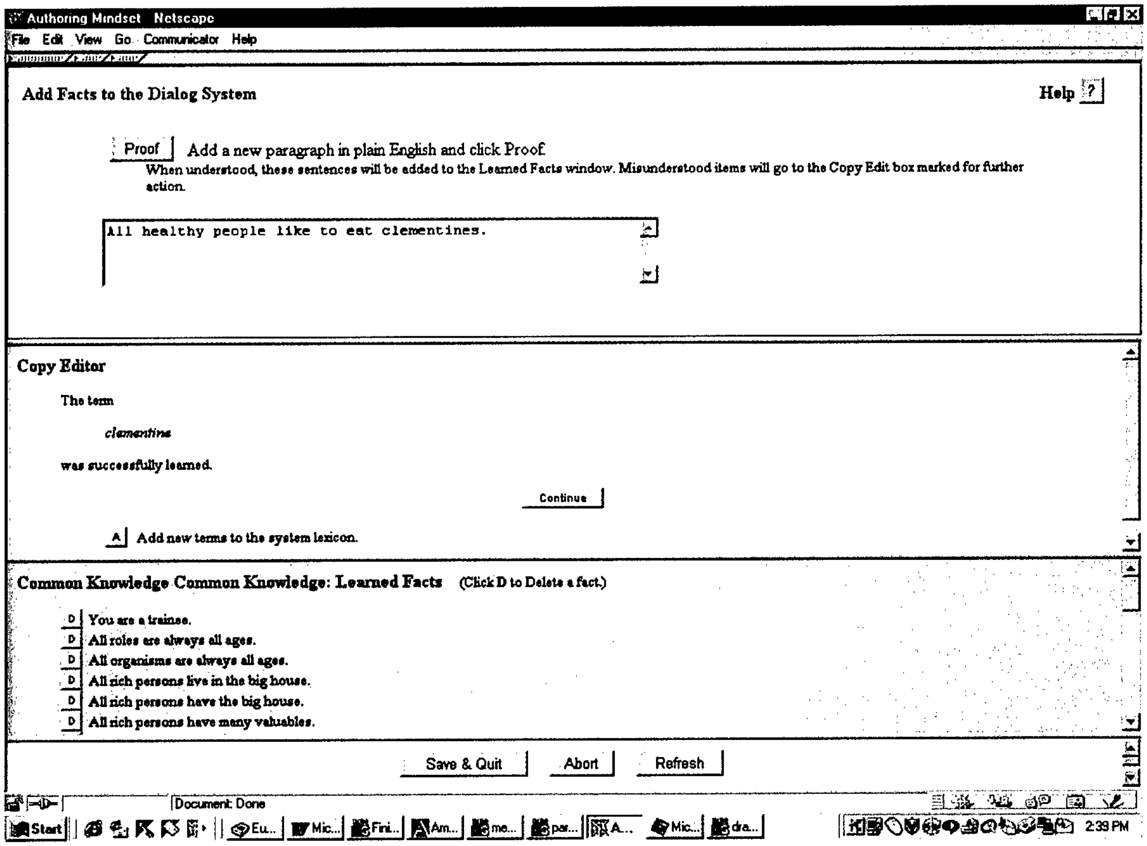
The top section is for adding facts to the knowledge base. The second section, **Add Words and Categories to the Dialog System** has the functions for extending the ontology and lexicon. Suppose we want to add *clementine* as a new kind of orange. We can enter *clementine* in the **Word** field, leave **Part-of-speech** as **Noun**, and **Relationship to** as **a kind of**, and then enter **orange** into the **Category** field.

But we can only do this if orange is a known category in the ontolexicon. Otherwise it will tell us the form contains an error and to please try again. To check that **orange** is a known category we can enter it in the bottom section and **Probe for a category**. The next GUI shows the part of the ontology that contains the category, orange, and one can then add *clementine* as a kind of orange.

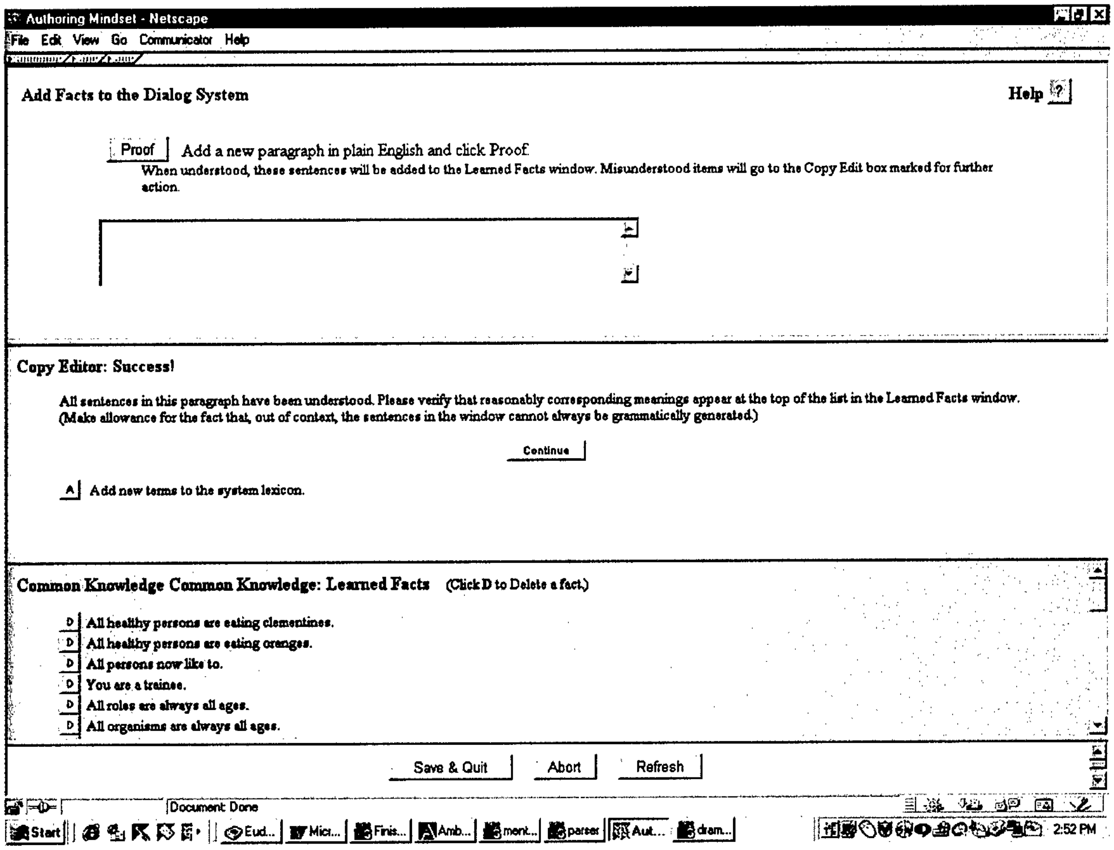


Adding new facts and rules

Once new terms are added they can be used in factoids. Now we can add a rule that "All healthy people eat clementines." We add the rule to be proofed for Common Knowledge:



The sentence "All healthy people like to eat clementines." is proofed (represented internally) as "All healthy persons are eating clementines." The results are shown at the top of the next page.

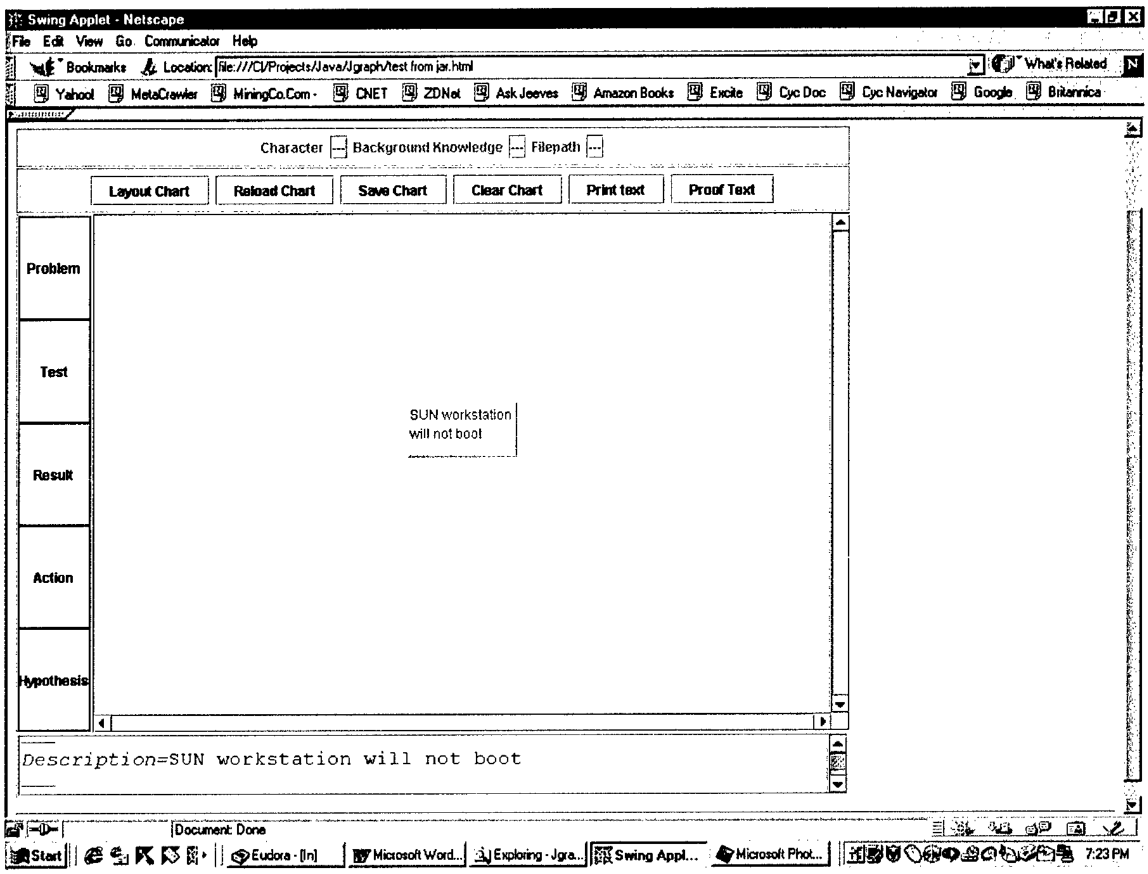

Authoring Mindset - Netscape
File Edit View Go Communicator Help
Add Facts to the Dialog System
Help
Proof
Add a new paragraph in plain English and click Proof.
When understood, these sentences will be added to the Learned Facts window. Misunderstood items will go to the Copy Edit box marked for further action.
Copy Editor: Success!
All sentences in this paragraph have been understood. Please verify that reasonably corresponding meanings appear at the top of the list in the Learned Facts window. (Make allowance for the fact that, out of context, the sentences in the window cannot always be grammatically generated.)
Continue
A Add new terms to the system lexicon.
Common Knowledge Common Knowledge: Learned Facts (Click D to Delete a fact.)
D All healthy persons are eating clementines.
D All healthy persons are eating oranges.
D All persons now like to.
D You are a trainee.
D All roles are always all ages.
D All organisms are always all ages.
Save & Quit
Abort
Refresh
Document: Done
Start
2:52 PM

Authoring an Effective Problem Space

Defining a new problem space

A new problem space is defined by specifying what kinds of problems the EPS handles. This problem space node is used as the root of the new EPS. New nodes can then be dragged from the palette of node types on the left onto existing nodes. Dialog boxes ask additional information required for the kind of node, for example, "What is the query?" for a test node.

The user clicks right on the Problem Description node and selects **Edit Node,** and enters "SUN workstation will not boot".



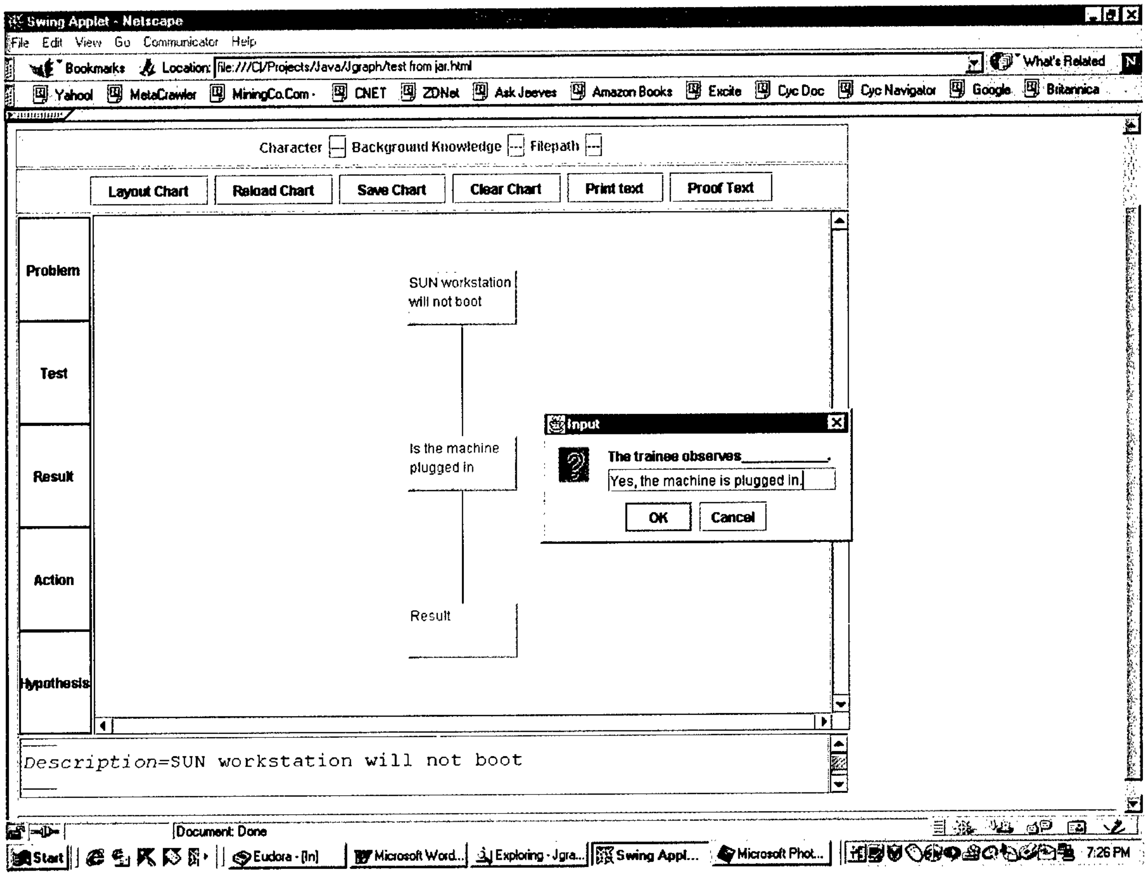
Note the palette of 5 different node types on the left. Any can be dragged over onto another node on the graph, although there are constraints to prevent errors when formulating an EPS. For example, dragging another problem space node or a test result node to be placed under the current problem space would be rejected, whereas a test node, action node, or hypothesis node will be accepted.

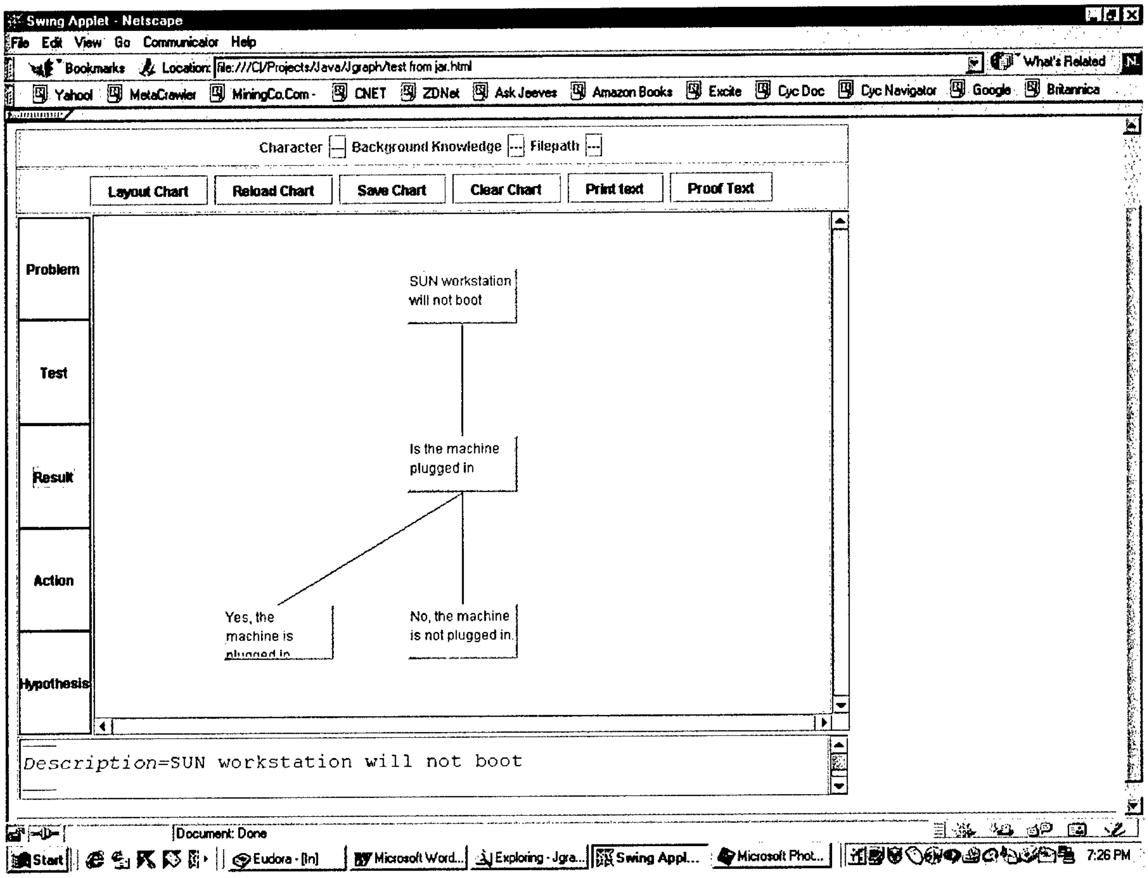
In this example the next node to be added is a Test node, as shown on the next screen. Pop-up dialog boxes prompt for node specific information. Pop-up dialog boxes prompt for text with fill-in-the-blank queries to encourage text wording that can be used in complete sentences and can be more easily proofed by Mentor.

In the next two screens two test result nodes are added. Possible observations corresponding to each result are stored with each test result node. Mentor will compare the student's answer to all possible test results to decide which branch to follow next. The final screen shows the EPS used for the SUN will-not-boot troubleshooting domain.

Editing a Problem Space

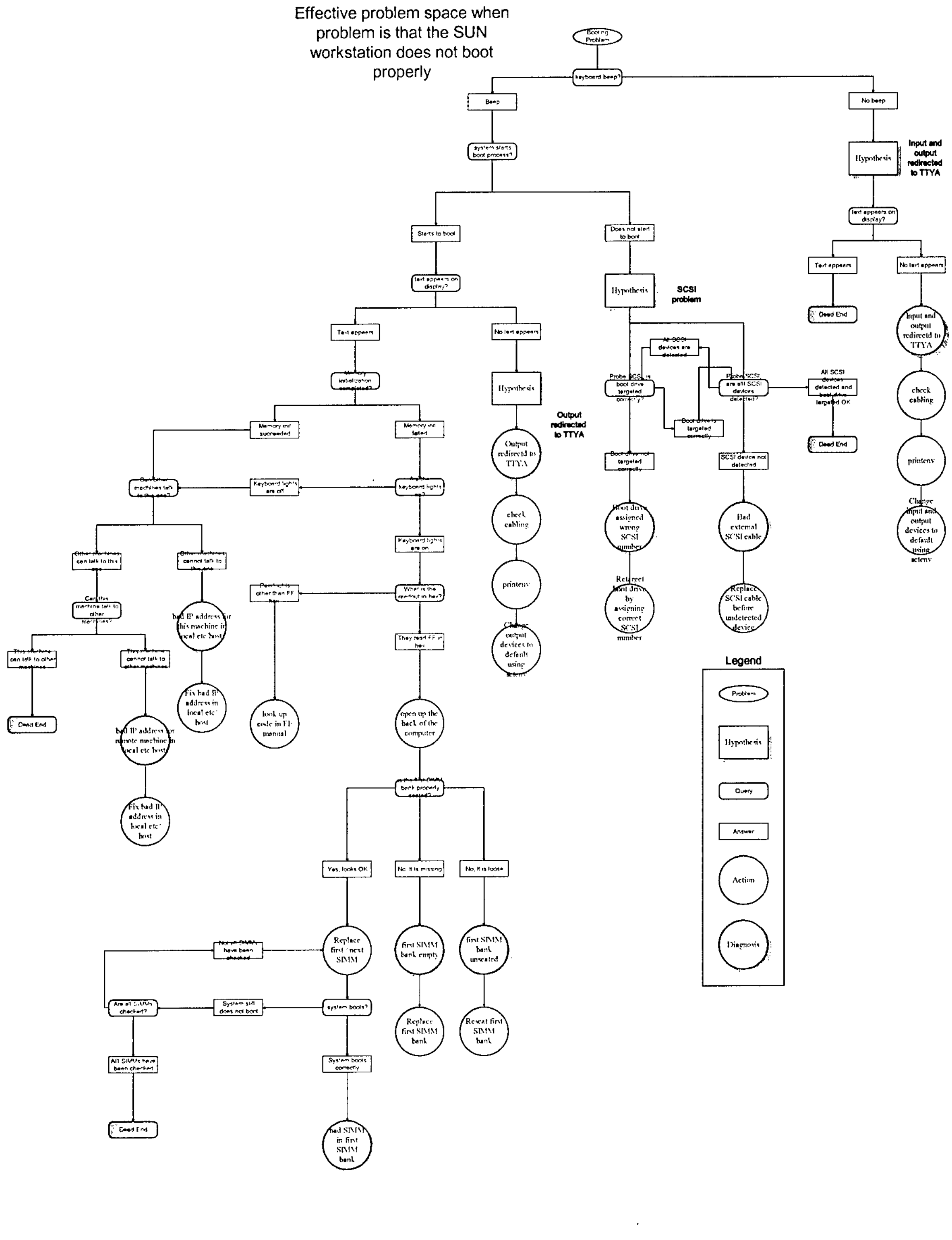
Any node in the EPS GUI can be right-clicked to bring up a menu of operations. If the node is a terminal node it can be deleted from the tree. The information specific to the node, including its name, can also be changed. In this way nodes can be removed from the tree, renamed, or have their content (e.g., query text) changed.




Swing Applet - Netscape
File Edit View Go Communicator Help
Bookmarks Location: file:///C|/Projects/Java/Jgraph/test from jar.html
What's Related
Yahoo! MetaCrawler MiningCo.Com CNET ZDNet Ask Jeeves Amazon Books Excite Cyc Doc Cyc Navigator Google Britannica
Character Background Knowledge Filepath
Layout Chart
Reload Chart
Save Chart
Clear Chart
Print text
Proof Text
Problem
Test
Result
Action
Hypothesis
SUN workstation will not boot
Is the machine plugged in
Yes, the machine is
No, the machine is not plugged in.
Description=SUN workstation will not boot
Document: Done
Start
Eudora - [In]
Microsoft Word...
Exploring - Jgra...
Swing Appl...
Microsoft Phot...
7:26 PM

The Sun workstation effective problem space

The graph below shows the effective problem space used for the SUN will-not-boot problem space. The diagram shown is drawn in a different format from the EPS GUI's format as it was developed prior to the EPS GUI.



SAMPLE DIALOGS

This section provides examples of dialogs with characters created using the Mentor system.

A dialog with a conversational agent, Mammood

Create a new character, Mammood Hadji, male, with personality traits: Helpfulness=Cooperative, and Manner=Rude. The character's personality = surly taxi driver in DC knows stuff about DC, and a short description of the problem domain = where to eat in DC.

Enter these factoids into Mammoods belief set:

> You are a taxi driver.
>
> You hate Washington DC.

Enter these factoids into Common Knowledge:

> All dogs chase cats.

Finally add Fido as an individual dog using the ontolexicon tools. Add the word "Fido" as a noun using the "is individual" relation to category "dog".

Here is the resulting conversation with Mammood:

> Bill: Do you like DC?
>
> Hadji: No, I do not like DC, dummy.
>
> Bill: Do you hate DC?
>
> Hadji: No, I do not hate DC.
>
> Bill: What is Fido?
>
> Hadji: Fido is one of the canines.
>
> Bill: What does Fido chase?
>
> Hadji: I wouldn't say it that way.
>
> Bill: What kind of animal does Fido chase?
>
> Hadji: I wouldn't say it that way.
>
> Bill: Does Fido chase dogs?
>
> Hadji: I wouldn't say it that way.
>
> Bill: Does Fido chase cats?
>
> Hadji: I wouldn't say it that way, you idiot.

A dialog with a conversational agent, Monty

We will create a new polite, cooperative, conversational character with title "General", first name, "Monty", and last name "Montgomery". For "Please give a short description of this character's personality:", say "Forceful". And for, "Please give a short description of this problem domain:",say "Tank command".

Tell him:

You are a general.
You command 40 tanks.
You have a Japanese sword
You have a British army
You are in Africa
You are reading a book on military strategy.

Here is the resulting conversation with Monty:

Bill: Where are you?

Monty: I am in Africa.

Bill: What are you reading?

Monty: I am reading a book.

Bill: What do you have?

Monty: I have a sword.

Bill: Do you have a British army?

Monty: Yes, I have an army, sir.

Bill: Do you like Germans?

Monty: No, I do not like any Germans.

Bill: Do you like British people?

Montgomery: No, I do not like any British people.

Bill: Where is your army?

Monty: I really don't know.

Bill: What is it about?

Monty: I don't know about what a book is.

Bill: Does your book discuss strategy?

Monty: It doesn't matter if my book discuss a strategy, sir.

Bill: What kind of sword?

Monty: I don't understand what.

Bill: What kind of weapons do you have?

Monty: I have no kind of weapons, sir.

Bill: Do you have any tanks?

No, I do not have any tanks.

Bill: Are you in the army?

Monty: No, I am not in the army.

Bill: Are you a soldier?

Monty: No, I am not a soldier.

Bill: What is a general?

Monty: I am a man, sir.

Bill: Do you like Englishmen?

Monty: No, I do not like an Englishman.