The RAND intelligent terminal agent (RITA) as a network access aid

by ROBERT H. ANDERSON and JAMES J. GILLOGLY
The RAND Corporation
Santa Monica, California

ABSTRACT

An operational “intelligent terminal agent” system called RITA is described. RITA uses production systems to store heuristics about dealing with the interactive protocols of external information systems. The use of production systems allows RITA to operate in either a pattern-directed or goal-directed manner. By creating new production rules as part of their operation, RITA agents can exhibit learning. Advantages and disadvantages of production systems for creating intelligent terminal agents are discussed, and an annotated transcript of a session with a RITA agent is given to illustrate its ability to aid a user in handling File Transfer Protocol on the ARPANET.

INTRODUCTION

Until recently, most users of information systems have been either “computer sophisticates”—such as computer programmers—or else users of systems, such as airline reservation systems, with a very limited set of options. However, with the continuing rapid decline in the cost of computer hardware and data communications, many interactive computer-based information systems are becoming cost effective for much broader categories of users. These newer systems will greatly expand the number of people interacting with computers in their daily activities, and will give access to a complex variety of interactive protocols, interfaces, command languages, and remote computing systems. People are going to need assistance in tailoring this variety of options to their specific needs and especially in freeing them from routine interactions and protocols which are not directly relevant to the content of their task.

We assume the complexities of dealing with the interactive protocols of computer networks are familiar to this audience, and are sufficiently documented† not to require further elaboration. This paper describes one possible solution to the problem of complexity: user access to computer networks aided by an “intelligent terminal agent.”

What is an intelligent terminal agent and why is it useful? The answer involves many aspects of the way people interact with computer systems as well as new alternatives becoming possible through advances in both hardware and software technology. The following observations concerning man/machine interaction and its supporting technologies form the basis for our design of such an agent:

1. Projected computer hardware cost trends and advances in microprocessor technology make it extremely likely that interactive computer terminals can be produced within five to seven years containing processing power and data storage equivalent to a present-day minicomputer, at a cost which is reasonable, assuming fairly intensive use of dedicated terminals by professionals as part of their job.

2. It is important to have certain information storage and handling capabilities locally*—most likely within the terminal itself—e.g., to provide “instantaneous” response to simple text manipulation commands and to simple error conditions.

Once local computing power becomes available within a terminal, it can be used to aid in interfacing with external information systems, such as the ARPANET or New York Times Information Bank, where much of the interactive protocol involves supplying standard responses which are not directly relevant to the task being accomplished. It is possible in the system described in this paper to teach an intelligent terminal to deal with such interactive protocols automatically, including instructions on dealing with certain error conditions, so that these details need not be remembered and handled manually by the user.

In addition, intelligent terminals should allow the user to define and set in motion “user agents”. Such an agent could:

Look at a calendar of events and start up services for the user automatically at certain times and dates. By manipulating calendar items, the human manager can progressively modify the plan

* “Locally” is used here to mean computing power and storage dedicated to the individual user and accessible via a very-high-bandwidth link—e.g., sufficient to rewrite a 3,000-character CRT display in 0.5 seconds.

† From the collection of the Computer History Museum (www.computerhistory.org)
being executed by the machine. For example, by changing the due date of a report, the schedule will be automatically altered for reminders and follow-up queries to be transmitted to persons making contributions.

Monitor the occurrence of various types of events, such as the arrival of a certain piece of network "mail", or the occurrence of a certain datum in a changing data base.

Deliver "interactive letters" to other users' terminals; these letters are capable of carrying on a dialog with the recipient, while in the process extracting information from him in a standard format suitable for further automated processing. Reference 2 contains an example of an interactive letter.

Manage transactions between a number of computerized services distributed on a computer network, monitoring their successful accomplishment.

We believe the desirability of the above list of services is a compelling reason to explore the design of intelligent terminal agents capable of running in present-day minicomputers. The remainder of this paper describes the design and operation of a system—the Rand Intelligent Terminal Agent (RITA)—which has been developed at Rand to meet all of the above requirements. RITA is currently operational on a PDP 11/45 minicomputer running the UNIX operating system. The next section discusses "production systems" as the software technology underlying RITA. The third section contains examples of RITA's operation as an interface to computer networks. The fourth section concludes with a discussion of RITA's implementation status and some remaining research questions which have been raised by our work to date.

PRODUCTION SYSTEMS AS THE BASIS FOR A TERMINAL AGENT

The basis for our design of a system meeting the above requirements is the use of production systems. A production system consists of a set of production rules (having a pattern part and an action part), which operate upon a data base (which we call a context), according to the actions of a rule interpreter, or monitor.

For example, two production rules might be:

Rule 1

IF: there is a message whose status is "awaiting action" and the identification-field of the message is not in the action-items of the user

THEN: put the identification-field of the message into the action-items of the user;

Rule 2

IF: the latest-command of the user is "show action items" and the state of the system is "command unfulfilled"

THEN: send the action-items of the user to the user and set the state of the system to "command fulfilled".

These rules would be part of a larger set of rules governing a message-handling user agent. They might be interpreted by a monitor that continually tests the "if" conditions in each rule of the set, and executes the "then" actions in any rule whose conditions are all true. Assuming messages with various attribute values, such as an identification field and status, are placed in the data base by some external (and possibly asynchronous) process, the above rules would update a list consisting of the identification numbers of all messages awaiting action, and show that set to the user upon his request. Other rules would themselves, or permit the user to, take other actions and as a consequence change the status of the messages and remove their identification number from the set of action items.

There are a number of interesting options in the design of production systems. For example, production rules can be used in either a goal-directed or pattern-directed manner. A goal-directed system has a designated goal, and the objective is to execute rules whose actions help to achieve that goal. These two modes of operation are discussed further under the heading "Monitors," below.

A good discussion of design options in production systems is contained in a recent survey article by Davis and King. That reference, as well as Reference 2, can be consulted for more details. A series of articles on the MYCIN system by E. H. Shortliffe and associates at Stanford University describes a particular goal-directed production system which has significantly influenced the design of the RITA system. Our design decisions in creating a production system for our particular needs are discussed below under four headings: data base, rules, monitors, and system architecture.

Data base

The data base upon which RITA rules operate is called a context; it consists of an unordered set of objects. Each object has a name, or type, and there can be more than one object in the context of the same type. There is neither an external structure imposed on the set of objects in the context nor a requirement that each object have a unique identifier associated with it. Each object can have one or more named attributes, and all attributes attached to an object must have names which are mutually distinct. Each attribute has an associated value, which is either a character string or an ordered list of values.
Objects, attributes, and values may be created or deleted dynamically by the actions of rules. If an attribute being tested by a rule's predicate does not exist, it is considered to be "not known." It is possible by a rule action (except within a goal-oriented monitor) to reset an attribute having a value back to the "not known" status. Goal-oriented monitors may not reset the value of any attribute; they may only set values which were previously not known. This restriction is necessary to preserve the integrity of information upon which chains of logical reasoning are based.

As an option, it is possible to attach a "level of certainty" to a string attribute value as it is being set. In this case, an attribute can have several different values associated with it, each with a different level of certainty. Levels of certainty are adjusted as additional positive or negative certainty factors for those values are asserted by the action of rules. Our use of certainty factors has been strongly influenced by their implementation in the MYCIN system, but differs in some details which will not be discussed here.

Figure 1 contains examples of object types and associated attribute names and values which might be used in a user agent within the RITA system.

The data structure we have chosen is not the most general one possible. As with other implementation decisions, we have chosen what we consider to be the simplest format and conceptual structure which allows the description of situations and heuristics related to intelligent terminal agents. With more experience in using the RITA system, some of these decisions are almost certain to change.

**Rules**

RITA rules are expressed in a finite syntax (technically, parsable by an LR(1) algorithm). We have chosen a syntax patterned after the specialized English

<table>
<thead>
<tr>
<th>object type</th>
<th>attribute name</th>
<th>sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>name</td>
<td>&quot;foo.baz&quot;</td>
</tr>
<tr>
<td></td>
<td>directory</td>
<td>&quot;jjg&quot;</td>
</tr>
<tr>
<td></td>
<td>site_id</td>
<td>&quot;rand-isd&quot;</td>
</tr>
<tr>
<td></td>
<td>size</td>
<td>20000</td>
</tr>
<tr>
<td></td>
<td>owners_name</td>
<td>&quot;gillogly&quot;</td>
</tr>
<tr>
<td>site</td>
<td>id</td>
<td>&quot;rand-isd&quot;</td>
</tr>
<tr>
<td></td>
<td>operating_system_name</td>
<td>&quot;unix&quot;</td>
</tr>
<tr>
<td></td>
<td>machine_type</td>
<td>&quot;pdp-ll/45&quot;</td>
</tr>
<tr>
<td></td>
<td>guest_account_name</td>
<td>&quot;netguest&quot;</td>
</tr>
<tr>
<td></td>
<td>guest_account_password</td>
<td>&quot;netguest&quot;</td>
</tr>
<tr>
<td></td>
<td>known-user_set</td>
<td>(&quot;jjg,&quot; &quot;rha,&quot; &quot;reg&quot;)</td>
</tr>
<tr>
<td>known_person</td>
<td>name</td>
<td>&quot;gillogly&quot;</td>
</tr>
<tr>
<td></td>
<td>primary_site_id</td>
<td>&quot;rand-isd&quot;</td>
</tr>
<tr>
<td></td>
<td>primary_directory</td>
<td>&quot;jjg&quot;</td>
</tr>
<tr>
<td></td>
<td>primary_password</td>
<td>&quot;whampus&quot;</td>
</tr>
<tr>
<td></td>
<td>secondary_site_id</td>
<td>&quot;cmu-10a&quot;</td>
</tr>
<tr>
<td></td>
<td>secondary_site_directory</td>
<td>&quot;250x12&quot;</td>
</tr>
<tr>
<td></td>
<td>secondary_site_password</td>
<td>&quot;foo&quot;</td>
</tr>
</tbody>
</table>

**Figure 1—Examples of RITA object types, attributes, and values**

```
Left-hand-side predicate clauses
IF: the name of the system is "unix"
IF: the name of the system is the name of the desired system
IF: the name of the system is not known
IF: there is a response whose arrival_time is less than the max_expected_delay of the system

Right-hand-side action clauses
THEN: set the name of the system to "net access program"
THEN: set the valid_id_set of the remote_site to the id of every site whose id is known
THEN: deduce the guest_account_name of the remote_site
THEN: create a remote_site whose id is "cmu-10a"
THEN: receive the next line from the system_10_pipe as the value of the response
THEN: send "Which rule do you wish to see" to the user
THEN: return success
```

Figure 2—Examples of RITA rule clauses

output form generated to display MYCIN rules to a user. We believe that this syntax is simple enough to be read and written by a computer-naive user. Figure 2 contains several examples of clauses which can be used in RITA rules; more complete examples are contained in the transcript in a later section.

Such facilities as string manipulation are provided in the RITA system by a set of primitive functions which may be called in the predicates or actions of rules.

**Monitors**

We have found that different types of monitors are necessary for various specific tasks and situations, and that no one monitor type is sufficient for our purposes. For example, interactions with an external information system to handle routine protocols are best handled by a LHS-scan monitor (one which tests the pattern part, or left-hand side, of rules against the context to determine the next rule(s) to be applied), acting in what might be called a "stimulus-response" mode. On the other hand, it is sometimes necessary for an intelligent terminal agent to make deductions (e.g., about the most likely site on the ARPANET for a particular person to have a mailbox, given that person's attributes). Deductions are best made by a RHS (right-hand side, or action part) scan, goal-driven monitor. In this form of monitor operation, one specific item of information (an attribute of an object) is designated as a goal. The monitor seeks to execute rules whose RHS set that attribute's value. To execute those rules, their LHS must be true when tested against the current context. If any such LHS is not true due to the lack of information about the value of some other object's attribute, that attribute becomes a (sub)goal of the
From the collection of the Computer History Museum (www.computerhistory.org)

The syntax module contains facilities for compiling symbolic form rules and data descriptions into an internal list-structure form. The monitor module accepts these "compiled" rules and has facilities for decompiling internal forms back into symbolic form upon request. It uses one of the available monitors to apply a rule set to a context, and emits trace information to a history file for use by diagnostic and tutorial facilities.

Advantages and disadvantages of production systems

Why were production systems chosen as the basis for the RITA system? The following advantages are often cited as accruing from their use. We have listed them in what we believe is an approximate decreasing order of importance for the particular application for which RITA was designed: namely, the construction of intelligent terminal agents.

1. Their explanatory capability.

Production system rules are intended to be modular chunks of knowledge and to represent primitive actions. Thus, explaining primitive acts should be as simple as stating the corresponding rule—all necessary contextual information should be included in the rule itself. Achieving such clear explanations, however, evidently strongly depends upon the extent to which the assumptions of modularity and explicit context are met.

The interested reader is referred to the MYCIN literature for an excellent example of the degree of explanatory power that can be achieved through careful design and implementation.

2. Simple control structure.

Due to the simple control structure of production systems, especially of the LHS scan type, we can imagine the following type of instructions being nearly sufficient to introduce a user to the operation of his terminal:

This terminal operates according to a set of rules. Whenever it finds a rule that is true, it applies that rule. If you want to know why it is asking you for some item of information, or why it took some action, type "why?" and it will show you the rules it followed in taking that action.

If, in addition, the rules themselves are in simple English so that they are directly readable by a user, then we believe he will find the operation of this device quite understandable. Although the user will of course not understand all the nuances of its operation, he is at least not bewildered at the start, and can add incrementally to his understanding with experience. The user must realize, however, during this initial introduction to the system that there are nuances and that he should not be overly complacent or trusting of system behavior.
A RHS scan backward-chaining system, although more complex in its control structure, can give rational explanations of its behavior in a manner that makes the flow of control among rules understandable.

3. Incremental addition of knowledge.
With proper design, production systems can allow gradual, incremental addition of knowledge and heuristics in a top-down manner. If the set of rules is unordered, then new rules can be added to the set without concern for their placement. A particularly appropriate time for the addition of new rules to a system is when the system, in a goal-oriented mode of operation, has asked a question of the user. A possible user response is to give the system a rule for determining that item of information from other information it has; upon receipt of that rule, the system will no longer ask that question, since it can now form a subgoal by backward-chaining through the new rule. In this manner gradual evolution of the behavior of the system takes place to meet the needs of the user in his possibly unique environment.

4. Trainability and learning.
Assume production rules are stated in a constrained syntax so that their meaning is understandable by machines, and that each rule is, to the extent possible, a "noninteracting chunk of knowledge or behavior." It then becomes possible for a computer program to create rules in the proper format and insert them into existing sets of rules to change the behavior of a production system. For examples of such adaptivity in production systems, see References 10 and 11.

There are also some disadvantages in the use of production systems. The two major ones are:

1. It can be difficult to code an operation in the form of a production system, particularly for goal-oriented rule sets. Considerable thought must be given to the choice of objects, attributes, and values by which a problem area is represented. (However, the problem of choosing a good data representation is certainly not unique to production systems; the problem lies more in trying to fit all applications into this particular procrustean bed.) One must also carefully choose certain attributes of objects to represent "state variables" which encode the state of a computation or deduction. The values of these state variables are tested by various rules to trigger their potential applicability. In this manner, production systems encode explicitly that which in ordinary high-level programming languages is implicit in the nesting of control statements. For example, a traditional nested control structure such as:

```
if A then
  if B then C
  else if D then E; else;
else G;
```

might be encoded in a production system in the following manner:

```
if A then state_1;
if state_1 and B then state_2;
if state_1 and not B then C;
if state_2 and D then E;
if not A then state_1;
if state_1 and B then C;
if state_1 and not B then C;
if state_2 and D then E;
if not A then G;
```

Such explicitness in a production system allows the desired relative autonomy of individual rules, but at the price of requiring the programmer to create names for many intermediate states of his process.

In the RITA system, we hope to overcome this disadvantage by having system experts create initial systems and user agents having general applicability. Individual users are expected, at least initially, to make only rather minor modifications and enhancements to the basic system. Therefore, the vocabulary and overall design of a user agent will be established, providing many guidelines and examples for the individual user.

EXAMPLE OF RITA OPERATION AS A NETWORK ACCESS AID

The following is an annotated transcript illustrating the operation of a RITA user agent designed to assist in file transfers from remote sites over the ARPANET. The agent consists of 42 rules and a context having 14 objects. Several representative rules from this user agent are shown in the middle of the transcript. Annotations are indented and enclosed in square brackets. User interactions with the agent are shown italicized.

```
% file-agent

["%" is the UNIX system prompt. An executable UNIX file named "file-agent" contains commands to start a RITA agent with the rules to be read from a file called "ftp.rules"].

UF: V21 Dec. 16
PARSER: V22 Dec. 8
MON: 17 Dec 75

[These three header lines are displayed by various RITA processes as they initialize]
ftp.rules:

*[Displayed by RITA to show this rule set is being loaded; any syntax errors, etc. would be printed at this time.]*

* run; *

[“*” is the RITA user front-end prompt; the “run” command starts the agent.]

*[The following 6 questions are generated by RITA during its backward-chaining deduction process.]*

What is the name of the current-file?

draft2

What is the site-name of the current-file?

sumex

What is the host-id of the current-file?

[The file is being retrieved from a site that the agent has never encountered before, so it checks that the site-name given is really the official host id for that site. As will be seen below, it doesn’t need to be told this again in a subsequent session.]

sumex-aim

What is the account-name of the current-file?

kowalski

What is the password of the current-file?

qwerty

What is the man-number of the current-file?

why

[At this point, the user doesn’t know what data is being asked for, so he uses the reserved word “why” to ask the agent for the chain of logic that led it to ask this question.]

That’s what I was supposed to deduce.

What is the man-number of the current-file?

[This is RITA’s answer if it has executed the action clause “DEDUCE man-number OF current-file” in some rule, but no goal-oriented rules are relevant in helping it deduce that; therefore, there is no deeper chain of logic that got it here. The user wants to better understand how the man-number is used by the agent, so he interrupts the agent’s operation and returns to its command level.]

* display all goals that test the man-number of the current-file; *

[This question lets him see all goal-oriented rules (denoted by a prefix “GOAL”) that test that attribute in their “if”, or predicate, part.]

GOAL not-known:

IF: the man-number OF the current-file IS NOT KNOWN OR the man-number OF the current-file IS " "

THEN: SET the file-prefix OF the current-file TO " " & SET the file-suffix OF the current-file TO " ";

GOAL before:

IF: nsubstr (0, 1, the man-number OF the current-file) IS "<"

THEN: SET the file-prefix OF the current-file TO the man-number OF the current-file & SET the file-suffix OF the current-file TO " ";

GOAL after. :

IF: nsubstr (0, 1, the man-number OF the current-file) IS NOT "<"

THEN: SET the file-suffix OF the current-file TO the man-number OF the current-file & SET the file-prefix OF the current-file TO " ";

[Having seen the above rules, the user thinks the system doesn’t handle TENEX systems correctly, and decides to give it a rule which will allow it to deduce the man-number of a file from the account-number on such systems. He therefore reinitializes the agent and types in a new rule at the agent’s command level.]

* restart; *

* goal tenex

 if: the type of the system is “tenex”
 then: set the man-number of the current-file to concat ("<", the account-name of the current-file, ">") ;
[He then re-runs the agent to see how it will act with this new rule added.]

* run;

What is the name of the current-file?
draft2

What is the site-name of the current-file?
sumex

What is the host-id of the current-file?
sumex-aim

What is the account-name of the current-file?
kowalski

What is the password of the current-file?
qwerty

What is the type of the system?
tenex

[Note that instead of asking for the man-number of the current-file, the system asked for the system’s type instead, through use of the newly-added rule.]

Will now attempt to retrieve file <kowalski>draft2 from the sumex-aim system

Sent: ftp

[Lines prefixed with “Sent:” and “Received...” are generated by the agent during its operation to show its interactions with the remote system on behalf of the user. The agent could easily be modified to be less verbose.]

Received from net: Host:

Sent: sumex-aim

Received from ftp: Connections established.

300 SUMEX-AIM FTP Server 1.44.0.0—at SAT 3-JAN-76 14:22-PST

[“>” is the ftp prompt character]

Sent: user kowalski

Received from ftp: 330 User name accepted. Password, please.

> 

Sent: pass qwerty

Received from ftp: 230 Login completed.

> 

Sent: retrieve <kowalski>draft2 ftpdata

[Note that this RITA agent automatically places retrieved files in a local file named “ftpdata”; through the addition of one rule, the agent’s behavior could be changed so that it requests a recipient file’s name from the user.]

File has been transferred

Received from ftp: 255 SOCK 3276899843

250 ASCII retrieve of <KOWALSKI>DRAFT2;1 started.

252 Transfer completed.

Sent: bye

Job finished done

% [All ARPANET File Transfer Protocol interactions necessary to retrieve the desired file were successfully accomplished automatically by the RITA agent.]

[The next day, the user wants to retrieve that same file again, since he just received an ARPANET message stating that he should look at some late revisions. This gives us an opportunity to see how much this particular RITA agent has learned from the previous session.]

% file-agent

UFE: V21 Dec. 16

PARSER: V22 Dec. 8

MON: 17 Dec 75

ftp.rules

* run;

What is the name of the current-file?
draft2

What is the type of the system?
tenex

Will now attempt to retrieve file <kowalski>draft2 from the sumex-aim system

[Note that once it received the file name “draft2”, it assumed all other attributes about the file from information received during the current session. All of this adaptive behavior is handled by several RITA rules which build new rules and data objects as their actions, then save the revised rule set on a disk file.]
To tune this agent's operation better, it should also be given a rule telling it that the type of a system doesn't change; once it knows the type of a remote system, it can assume that it will stay constant. With that information, the agent would stop asking for the type of previously encountered systems.

Sent: ftp
Received from net: Host:
Sent: sumex-aim
Received from ftp: Connections established.
300 SUMEX-AIM FTP Server 1.44.0.0—at SAT 3-JAN-76 14:26-PST
>
Sent: user kowalski
Received from ftp: 330 User name accepted. Password, please.
>
Sent: pass qwerty
Received from ftp: 230 Login completed.
>
Sent: retrieve <kowalski> draft2 ftpdata
File has been transferred
Received from ftp: 255 SOCK 3276899843
250 ASCII retrieve of <KOWALSKI>DRAFT2;1 started.
252 Transfer completed.
>
Sent: bye
Job finished done

* exit;
exiting.
%

% file-agent
UFE: V21 Dec. 16
PARSER: V22 Dec. 8
MON: 17 Dec. 75
ftp.rules
* run;

What is the name of the current-file?
draft3

What is the site-name of the current-file?
sumex

What is the type of the system?
tenex

Will now attempt to retrieve file <kowalski> draft3 from the sumex-aim system

[Note that this time it asked for the site-name, but once it received a previously known response, it automatically retrieved other needed attributes about that site, such as its formal ARPANET host id. As mentioned above, the “type of system” question is unnecessary, and should be eliminated through the addition of another rule.]

Sent: ftp
.
.

[We omit the remainder of this session transcript, since it proceeds in the same manner as the interactive protocols shown above.]

CONCLUSION

This paper has discussed the design of the RITA system. Within RITA, user agents consisting of sets of production rules can be created to operate either autonomously or in interaction with a user. One major application of such user agents is to aid in interactions with computer networks and remote information systems. RITA is currently operational on a PDP 11/45 minicomputer running under the UNIX operating system. It occupies about 58K bytes of core, allocated among two separate processes. User agents comprising about 50 rules are now in use, and can, for example, handle ARPANET file transfer operations and interactions with the New York Times Information Bank.

This is, however, a report on work in progress. Some of the questions that remain unanswered at this stage of the research project are:

- Will a computer-naive user really be able to modify the operation of a user agent by adding or modifying rules? If so, how long a prior familiarization period is required?
- At what level of size or complexity of a user agent will speed of operation and efficiency become important considerations?
- What about system security? Knowledge about passwords, access keys, data formats, and account numbers for external systems might well reside within RITA user agents, making the intelligent terminal itself a valuable target for compromise. Even with physical security, such as restricting access to the room containing the terminal, often

From the collection of the Computer History Museum (www.computerhistory.org)
there will still remain external communication paths to the machine that allow possible access to data. We need to understand more about the constraints which must be placed on access to intelligent terminals containing sensitive data in representative user environments.

We are encouraged by our initial experimentation in the use of production systems to represent heuristics governing intelligent terminal behavior. Their ability to provide explanations of that behavior, to be modified and incrementally extended by a user, and to operate in either a pattern-directed or goal-directed manner are all potentially valuable features. We believe the RITA system, whose design we have discussed here, provides a good testbed for the demonstration and evaluation of these intelligent terminal agent capabilities.

ACKNOWLEDGMENTS

Bob Greenberg designed and implemented many key features of RITA's user interface and syntax modules. The particular RITA agent whose operation created the transcript presented earlier was created by Dr. Don Waterman. The support and encouragement to date of our prior ARPA program manager, Dr. Craig Fields, and our current program manager, Steve Walker, are gratefully acknowledged.

REFERENCES


