Use of a Forth-Based Prolog for Real-Time Expert Systems

I. Spacelab Life Sciences Experiment Application

William H. Paloski, Louis L. Odette, Alfred J. Krever,† and Allison K. West*

KRUG International
Technology Life Sciences Division
17625 El Camino Real, Suite 311
Houston, TX 77058

Abstract

A real-time expert system is being developed to serve as the astronaut interface for a series of Spacelab vestibular experiments. This expert system is written in a version of Prolog that is itself written in Forth. The Prolog contains a predicate that can be used to execute Forth definitions; thus, the Forth becomes an embedded real-time operating system within the Prolog programming environment. The expert system consists of a data base containing detailed operational instructions for each experiment, a rule base containing Prolog clauses used to determine the next step in an experiment sequence, and a procedure base containing Prolog goals formed from real-time routines coded in Forth. In this paper, we demonstrate and describe the techniques and considerations used to develop this real-time expert system, and we conclude that Forth-based Prolog provides a viable implementation vehicle for this and similar applications.

Introduction

The recent flurry of activity in commercial expert system development has all but bypassed the real-time computing community. Although it may be desirable to incorporate expert systems in certain real-time computing problems, the amount of computing over symbols (reasoning) required by an expert system is difficult to implement in real time. Indeed, it is often difficult to implement the amount of computing over numbers required by real-time applications. The commercial artificial intelligence (AI) industry is moving slowly toward providing fully formed products for real-time expert system development (cf. [WOL87], [LEI87]); however, general real-time symbolic processing remains an AI research problem. It may be several years before AI languages and development/delivery environments can play a significant role in real-time applications.

Owing to this lack of suitable, commercially available real-time expert system development vehicles and to its interest in developing expert systems for data acquisition and control applications, the Forth community has recently become active in developing real-time expert systems [PAR86]. Some of the contributions of this community in this area include a diesel electric locomotive repair expert system [JOH83], an orbiting spacecraft command and control expert system [HAR86], a spacecraft trajectory processing data error detection expert system (TRAPS) [RAS86], a real-time polysomnographer expert system (FORTES) [RED87], a sleep disorder diagnosis system [TRE87],

*Applied Expert Systems, 5 Cambridge Center, Cambridge, MA 02142
†FORTH, Inc., 111 N. Sepulveda, Manhattan Beach, CA 90266

some general real-time expert system development packages (EXPERT-2 [PAR84], FORPS [MAT87]), and a real-time version of the common expert system development language OPS5 (REAL-OPS) [DRE86]. In each of these applications, a set of high-level (AI) programming tools (i.e., inference engine, language parser, etc.) is built from Forth primitives to take advantage of the high run-time execution speed offered by Forth.

In the current paper, we describe a new Forth-based real-time expert system. This application is being developed using a unique implementation of the logic programming language Prolog, in which the Prolog interpreter is embedded in a Forth environment [ODE87]. This Prolog is fully compatible with the de facto standard described by Clocksin and Mellish [CLO81] and provides a simple and direct interface to the underlying Forth. This interface allows creation of Prolog goals from Forth words and provides the basis for rapid development and integration of the real-time routines into an expert system knowledge base. Real-time algorithms stored in the knowledge base through this mechanism can subsequently be executed on a logic-driven basis by the expert system.

The real-time expert system presented here will be used to assist astronauts in performing a series of life sciences experiments aboard the First International Microgravity Laboratory (IML-1) Spacelab mission. It will serve as the operator interface to the experiment data acquisition and control system. The series of experiments to be controlled by the expert system is known as the Microgravity Vestibular Investigations (MVI) and is designed to study the role of the inner ear in space motion sickness. An expert system was considered for the MVI experiments because of the complexity of the experimental procedures, the suboptimal experiment environment provided by Spacelab, and the high cost of failure. Highlights of this application are presented to demonstrate our general approach to building real-time expert systems using the Forth-based Prolog.

Instrumentation

The MVI experiments comprise six separate scientific functional objectives, each of which is addressed by performing 3 to 24 related experiments. These experiments will be performed during the IML-1 mission by teams of two astronauts. One astronaut will serve as the experiment subject. He will be strapped into a rotating chair mounted in the Spacelab center aisle and will don a helmet supporting various physical and physiological sensors. Throughout the experiments, he will be presented with inertial, auditory, and/or visual stimuli, and his physiological responses to those stimuli will be monitored, graphically displayed, and transmitted to Earth. The other astronaut will serve as the experiment operator. He will be responsible for setting up and calibrating each of the sensors and for performing each step of the 65 experiments. During each experiment, he will also monitor the response, safety, and well-being of the subject and communicate with ground-based scientific investigators.

The experiments will be controlled by the astronaut/operator using a rack-mounted console known as the Experiment Control and Data Interface (ECDI; Figure 1). The heart of this console is an IBM PC-compatible, 8086-based, laptop microcomputer (Grid Case III, Grid Systems Corporation, Mountain View, California). Within the ECDI, the PC bus is extended to a 9-slot expansion chassis (IBUS Systems Corporation, San Diego, California) which houses all the data acquisition, control, and Spacelab system interface hardware. Data acquisition and control capabilities are provided by two analog interface boards (DT2801, Data Translation, Marlborough, Massachusetts) containing a total of sixteen 12-bit differential channels of analog-to-digital (A/D) conversion, four 12-bit channels of digital-to-analog (D/A) conversion, and 32 bits of digital input/output (I/O) lines. Analog data from electro-oculogram (EOG), head acceleration, and head position sensors attached to the subject and helmet will be sampled at either 128 Hz (5 channels) or 32 Hz (10 channels) and digitized using the A/D converters. Command signals, issued through the D/A converters and digital I/O ports, will control an optokinetic device, a light emitting diode (LED) array, ear phones, and other experiment stimulus-producing devices.

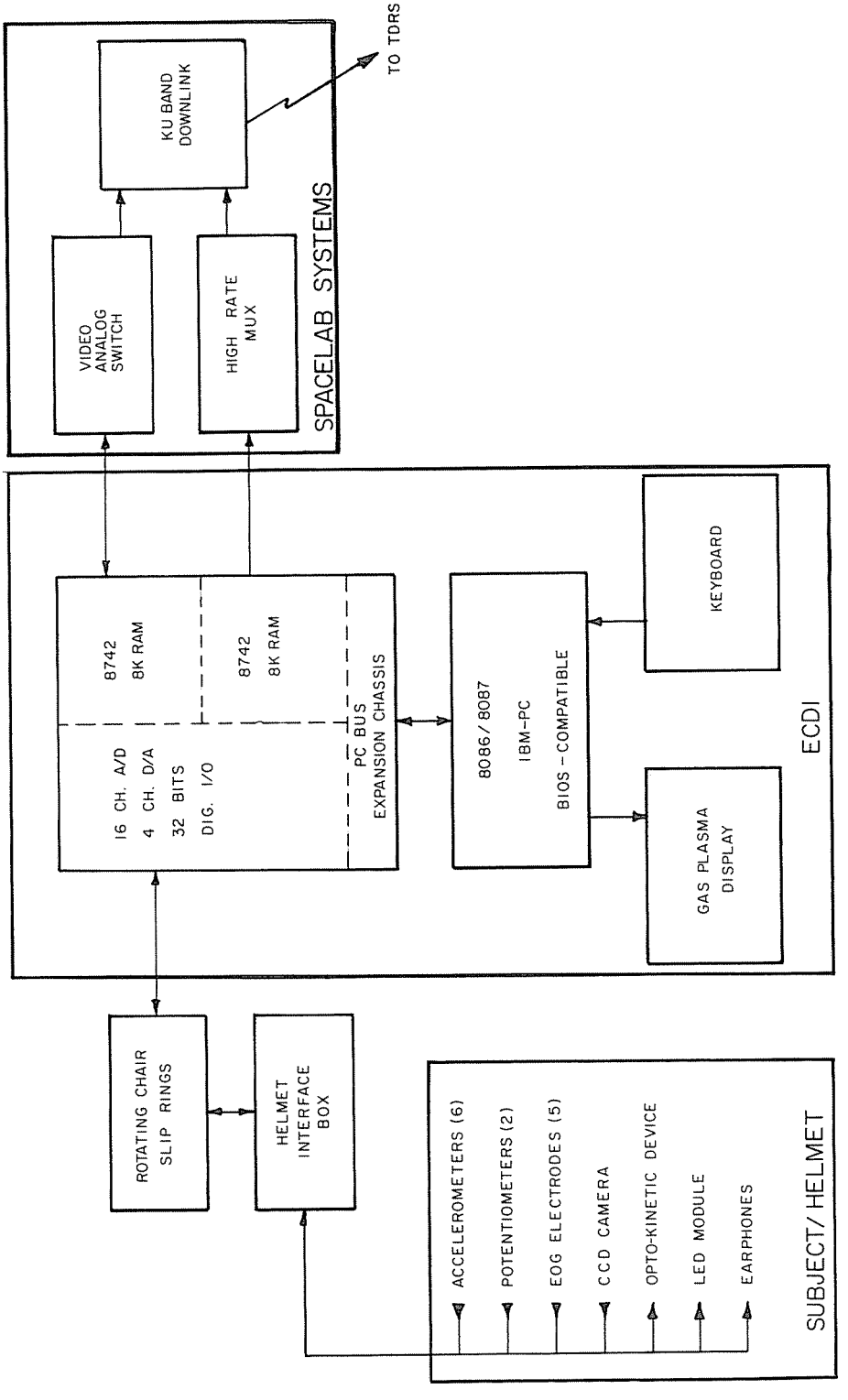

Figure 1
Block diagram of MVI hardware.

Following digitization, two channels of experiment sensor data will be graphically displayed in real time on the ECDI computer screen. In addition, all acquired data, as well as all command and experiment condition data, will be serially transmitted at 25.6 Kbits/sec to Spacelab downlink systems. Spacelab system interfaces are provided by two custom boards containing I/O-mapped, 8742 Universal Peripheral Interface processors. These boards will control data transfer between the experiment computer and either the Spacelab high rate multiplexer (HRM) or the Spacelab video analog switch (VAS) using pairs of memory-mapped RAM ping-pong buffers and Universal Asynchronous Receiver/Transmitters. The HRM and VAS provide interfaces for digital and video data telemetry to ground-based scientific and mission control observers.

Expert System

Although Spacelab provides a unique facility for studying the effects of microgravity on various physical and physiological processes, its environment is less than optimal for performing complex scientific experiments. Crew time and training are limited, communication between astronaut/operators and ground-based scientific investigators is limited, and crew members are likely to experience some degree of motion sickness during the first few days of a mission. The MVI expert system is designed to reduce the impact of these factors on the quality of the MVI data collection. This expert system should enhance the ability of the astronaut/operator to collect high quality data in the Spacelab environment by managing the complexity of the experimental procedures and by offering guidance through the experiment operations.

The heart of the expert system is its knowledge base, which is made up of a data base containing facts about experiment procedures and protocols, a rule base containing general rules for carrying out the experiments under normal and abnormal conditions, and a procedure base containing real-time routines coded in Forth. The rule base is used to step the operator through each experimental protocol, as specified in the data base, while simultaneously activating appropriate Forth tasks stored in the procedure base. The knowledge base is primarily declarative, relying on the Prolog inference engine for control; however, the Prolog **builtin** predicate, used to incorporate Forth routines into the knowledge base, allows procedures to be invoked from Prolog (a complete description of the Forth-based Prolog is provided in a companion paper in this issue [ODE87]).

A knowledge base structure was selected for representing the experiment procedure data rules in order to provide a flexible system that could be easily modified. This format allows separation of the data from the inference rules and real-time routines and thereby permits restructuring of rules without changing the data organization or real-time behavior. The structure of a knowledge base also greatly reduces the distance between the program and the user; both data and rules are stored in blocks of text that can be read as paragraphs by natural language processing routines as well as by the programmer and the expert Principal Investigators. This feature will aid in the development of user-friendly I/O routines and explanation facilities.

There is a top-level query language through which the experiment operator has access to the rule base. Upon being queried, the rule base extracts appropriate data from the data base and provides that data as input to real-time algorithms in the procedure base. The real-time algorithms then use this data in setup and execution of machine control tasks. At the present time, the data and procedure base development is essentially complete. The rule base remains under development; however, all of its essential components, including those described below, have been developed and tested. A detailed description of the knowledge base and execution process follows.

Data Base

The MVI expert system data base consists of six records, each of which encodes all of the operational requirements for a single functional objective. Each record has a tree structure (Figure 2). The primary node in each record is the functional objective identifier. Each of the n secondary nodes branching from the primary node identifies the location of all the information required to

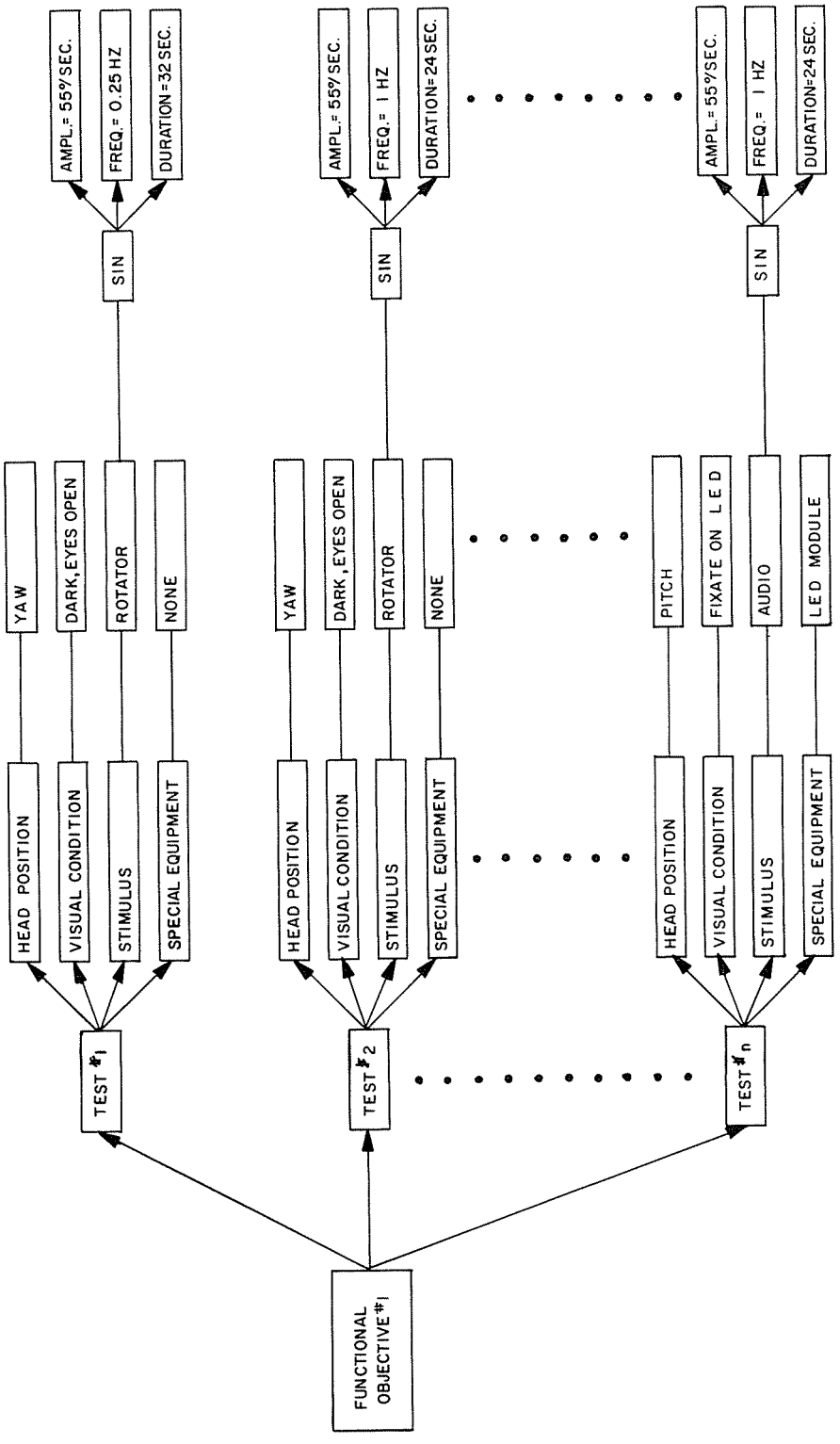

Figure 2
Example of the tree structure of the expert system data base.

perform a single experimental protocol (test) within the functional objective. Four tertiary nodes branching from each secondary node identify the location of the head position, visual condition, experimental stimulus, and special equipment information required by the particular protocol. The set of nodes branching from the tertiary nodes contains the actual data pointed to by the identifiers. All nodes further to the right add clarification and refinement to this data. As an example, Figure 2 shows that the experimental protocol Test #1 of Functional Objective #1 requires that an inertial stimulus be provided by the rotating chair and that this stimulus have a sinusoidal velocity profile with an amplitude of 55°/sec, a frequency of 0.25 Hz, and a duration of 32 seconds.

The Prolog clause required to encode a data tree similar to that diagrammed in Figure 2 is presented in Figure 3. Each branch node is represented by a predicate having an arity equal to the number of parenthetical objects following it and separated by commas. The arity of a predicate is fixed. For example, the arity of the predicate **functional_objective** is 3: the object **1** (the functional objective number); the object **vor_suppression** (the functional objective name); and a list, delimited by square brackets, which contains objects, predicates, and other lists that identify the remainder of the data required for Functional Objective #1. In Prolog, records can be organized into pseudo-English paragraph structures like that in Figure 3. This helps to bring the source code much closer to both the programmer and the domain expert. Prolog unification is the mechanism used by the expert system to access data fields within the data base.

```
146 LIST
 0      /* expert system development area */
 1      functional_objective(1,vor_suppression,
 2      [test(1,head_position(yaw),
 3              visual_condition(d,[dark,eyes_open,straight_ahead]),
 4              stimulus(rotator(sin(ampl(55),freq(25),dur(32)))),
 5              special_equipment(none),
 6       test(2,head_position(yaw),
 7              visual_condition(d,[dark,eyes_open,straight_ahead]),
 8              stimulus(rotator(sin(ampl(55),freq(1),dur(24)))),
 9              special_equipment(none)),
10       test(3,head_position(pitch),
11              visual_condition(f1,[fixate_on_LED_module_target]),
12              stimulus(audio(sin(ampl(55),freq(1),dur(24)))),
13              special_equipment(LED_module))]).
14
15
```

Figure 3

Prolog clause required to encode a data base record similar to the example presented in Figure 2.

Rule Base

The rule base will contain approximately 50 Prolog clauses, which the operator accesses via the user query language and the expert system accesses through unification. The primary execution rules have been developed, and some of them appear in Figure 4. Lines 2-3 in Block 147 present the general rule for executing a functional objective. The operator can execute a functional objective using the query **fo(X)**. If **X** is instantiated (equivalenced) to a functional objective number for which a data record exists within the expert system data base, then that particular functional objective will be executed. If **X** is instantiated to a functional objective number for which no data record exists, the second **fo(X)** clause (Block 147, Line 5) will be invoked to report the error. If **X** is not instantiated, the first functional objective record in the data base will be found and used for execution. Thereafter, successive functional objectives could be executed by forcing Prolog backtracking at the conclusion of each functional objective execution. This could be accomplished by requesting more solutions to the top-level query.

```
147 LIST
 0  /*  expert system development cont  */
 1
 2  fo(X) :- functional_objective(X,Y,L),!,
 3           execute_fo(L).
 4
 5  fo(X) :- nl,nl,write(invalid_fo_number),nl.
 6
 7
 8  execute_fo(L) :- member(test(N,HP,VC,ST,SE),L),
 9                   perform_test(N,HP,VC,ST,SE).
10
11
12  execute fo(L) :- nl,nl,write(fo_complete),nl.
13
14
15

148 LIST
 0  /*  expert system development cont  */
 1
 2  perform_test(N, head_position(Pos), visual_condition(Eyes,_),
 3              stimulus(Input_Device), special_equipment(Device)) :-
 4
 5                  set_up(Device),
 6                  restrain_head(Pos),
 7                  command_subject(Eyes),
 8                  data_on,
 9                  start(Input_Device),
10                  stop(Input_Device),
11                  data_off,
12                  command_subject(relax).
13
14
15
```

Figure 4
Prolog source code implementation of a portion of the MVI expert system rule base.

Procedure Base

The procedure base is a set of Forth colon and code definitions used to control all the MVI experiment hardware and to execute real-time data acquisition, display, and transmission tasks. Sample code demonstrating the mechanism used for building the procedure base is presented in Figure 5. Block 141 is a Forth block, compiled using the Forth `LOAD` command. Block 145 is a Prolog block (as are all blocks in Figures 3 and 4) and is compiled using the Forth word `CONSULTING` (see Block 141, Line 11), which has an action similar to that of the Prolog predicate **consult** [CLO81].

In Block 141, a small polyFORTH terminal task, `SIMUL`, is constructed in Line 1 [VAN83]. A simple function is assigned to that task: the Forth word `SEE` (Line 4) causes the contents of the variable `EVENTS` to be displayed on the user's terminal and increments the displayed value twice per second; the Forth word `DONE` (Line 5) deactivates the background task. In Lines 7 and 8, two

new Forth words, $DATA_ON and $DATA_OFF, are defined. These words are formatted to become Prolog built-in predicates; they include the R> DROP necessary for execution in the Prolog environment [ODE87] and a $TRUE word following all other execution, which indicates to Prolog that the word succeeds (is true). $DATA_ON starts the terminal task and $DATA_OFF stops the terminal task. The Forth words $DATA_ON and $DATA_OFF are defined as Prolog objects **data_on** and **data_off** in Block 145 (Lines 4 and 5). These objects can be used as Prolog goals, as can any other Prolog predicate; however, the process of testing one of these goals results in execution of its Forth procedure. Forth CODE definitions can be made Prolog objects using the same technique.

```
141 LIST
 0  ( Simple multitasking example)
 1  500 64 0 TERMINAL SIMUL     GILD   SIMUL CONSTRUCT
 2
 3  VARIABLE EVENTS
 4  : SEE    SIMUL ACTIVATE BEGIN 1 EVENTS +! 500 MS AGAIN ;
 5  : DONE   SIMUL ACTIVATE STOP ;
 6
 7  : $DATA_ON   R> DROP  ." Start" CR  SEE  $TRUE ;
 8  : $DATA_OFF   R> DROP  ." STOP" CR  DONE  $TRUE ;
 9
10
11  145 148 CONSULTING  ( application rule base follows)
12
13
14
15

145 LIST
 0  /* test screens for prolog -- needs SIMUL task */
 1  member(X,[X,_ ]).
 2  member(X,[_,Y]) :- member(X,Y).
 3
 4  data_on :- builtin($DATA_ON) .
 5  data_off :- builtin($DATA_OFF) .
 6
 7
 8
 9
10
11
12
13
14
15
```

Figure 5
Forth and Prolog source code used to create an example procedure base entry.

Logic-Driven Real-Time Procedures

The main advantage of this real-time expert system is its ability to perform real-time data acquisition and control tasks on a logic-driven basis. To perform Functional Objective #1 of the MVI experiments, for example, the operator could type in **fo(1)** at the ECDI keyboard. This would cause Prolog to locate the first **fo(X)** rule (Figure 4, Block 147, Line 2), instantiate the **X** to 1, and attempt to prove the rule true by satisfying all the goals on its right-hand side. To satisfy the final goal **execute_fo(L)**, Prolog would locate the first **execute_fo(L)** rule (Block 147, Line 8) and attempt to satisfy each of the goals on its right-hand side. The first of these goals, **member(test(N, HP, VC, ST, SE), L)**, identifies the parameters of the first test in the data base record associated with Functional Objective #1 (Figure 3) using the member rules given in Block 145, Figure 5. The second of these goals, **perform_test(N, HP, VC, ST, SE)**, causes Prolog to perform that test using the **perform_test** rule in Block 148. Prolog performs the test as a side effect of trying to prove the **perform_test** goal true. Unification is used to pass parameters found in the data base record into the procedure base.

The goals in the body of the **perform_test** clause sequentially perform a generic vestibular experiment. For example, using the Prolog execution logic (cf. [CLO81]) and the data base record presented in Figure 3, the variable **Device** would be instantiated to **none** in the **perform_test** rule. The first goal in the body of that rule would then become **set_up(none)**, indicating that no special equipment is required to perform this particular test. The operator would be informed of this fact when Prolog attempted to satisfy that goal (code not shown). As the Prolog execution continued, the operator would next be instructed to restrain the subject's head in the yaw position, then to be sure that the subject's eyes are in the dark, and finally to command the subject to keep his eyes open and look straight ahead (code not shown).

Next, having completed the setup phase of the test, the expert system would begin performing the test. To do this, Prolog would attempt to satisfy the goal **data_on**, which is a built-in Forth definition (described previously). The Forth definition `$DATA_ON` would be executed as Prolog attempted to determine whether the goal **data_on** is true (the last Forth word in the `$DATA_ON` definition is `$TRUE` which indicates to Prolog that the goal has succeeded). Thus, in satisfying the **data_on** goal, Prolog indirectly executes the Forth definition `SEE`. In the current example, `SEE` activates the background task `SIMUL` (see **Procedure Base**). In the MVI expert system, however, the **data_on** goal activates a Forth background task that acquires analog data from the subject, graphically displays selected data channels on the computer screen, and transmits the acquired data into the Spacelab downlink systems. Once the data acquisition, display, and transmission systems had been activated, the expert system would initiate the experiment stimulus. This would occur as Prolog attempted to satisfy the next goal in the body (**start(Input_Device)**) and indirectly activated, in the MVI expert system, another Forth background task, which would control that device according to parameters passed to the task from the data base. Once the stimulus provided by the input device was complete, the **start(Input_Device)** goal would succeed. The expert system would then turn off the device control background task as Prolog satisfied the **stop(Input_Device)** goal and the data acquisition task as Prolog satisfied the **data_off** goal. Finally, the operator would be notified that the test was complete and would be instructed to allow the subject to relax as Prolog satisfied the **command_subject(relax)** goal.

At this point, the expert system would have completed performing the experiment. Prolog would have satisfied **perform_test**, which, in turn, would satisfy **execute_fo**, which is the last goal of **fo(1)**. Prolog would therefore notify the operator that **fo(1)** was complete and would await further instruction. The operator would then have the option of accepting this solution (thereby concluding the experiment) or of asking Prolog to find a new solution, should one exist. The latter option would cause Prolog to backtrack (see [CLO81]) to attempt to resatisfy the initial goal. The Figure 4 rule base is set up so that backtracking would result in an attempted resatisfaction of the **execute_fo(L)** goal. Prolog would successfully resatisfy this goal if it found another test (one not

previously found) on the list of tests associated with Functional Objective #1. If such a test were found, it would be executed using the logic presented above (only the parameter values would change). Thus, following each test in the functional objective, the operator would be given the option of proceeding with the next test or aborting the study. By this mechanism, a single set of Prolog rules (Figure 4) can be used to carry out every test under all functional objectives. If the operator chooses to backtrack to the next test after all tests for a particular functional objective have been performed, the **execute_fo(L)** goal would fail. This failure would cause the **fo(1)** goal to fail and Prolog to await a new input goal from the operator.

Real-Time Operation

The MVI expert system provides real-time experiment control by activating and deactivating polyFORTH background and terminal tasks [VAN83] on a logic-driven basis. A portion of the MVI round robin multitasking loop is presented in Figure 6. Prolog (and the expert system) reside in the `OPERATOR` task, which is the primary terminal task in the round robin loop; all keyboard entries are handled through this task. When the MVI computer is turned on, the only active task is `OPERATOR`. The expert system controls the graphics display terminal task and the device driver background tasks as side effects of responding to operator queries (see the previous section). Only those tasks that need to be active at any point in the experiment protocol are activated. Once a task has completed its required action it is deactivated.

To maintain accurate, high rate (128 Hz) data sampling intervals, data acquisition is controlled by an interrupt service routine (ISR) triggered by a Spacelab clock signal. The data transmission background task is controlled by the ISR to assure synchronization between the MVI data stream and the Spacelab downlink system. The expert system can control data acquisition and transmission by masking and unmasking the Spacelab clock interrupt signal and by changing the ISR in use by altering its vector address. Either of these techniques can be employed on a logic-driven basis. By using this multitasking technique, various real-time processes can be executed concurrently with the expert system. This reduces the required logical inference processing rate of the Prolog.

Discussion

Forth is one of several possible development languages for implementing real-time data acquisition and control processes. We chose it because of its speed, small size, and ability to support rapid prototyping. Prolog is one of several languages for symbolic computation. We chose to use it for several reasons. One significant factor was our familiarity with it. We have several years experience building significant expert system shells and applications (cf. [BAR85]); by implementing Prolog we could reuse a good portion of this software. Another important factor was that Prolog has a small standard kernel; thus, its implementation could be made small and could be accomplished quickly. In our estimation, this compactness is in contrast to LISP; few have braved attempts at a full implementation of the Common LISP standard. Another selection factor was that there is a large community of Prolog users and programmers as well as a sizable body of literature describing Prolog solutions to problems in symbolic computation. The availability of this Prolog literature will allow us to avoid both the reinvention of the wheel and the problems inherent in translating programs. The popular production rule language, OPS5, shares some of these features. Indeed, Dress has developed a real-time version of OPS5 using Forth [DRE86]. Unfortunately, the data-directed style of OPS5 was not a good fit with the goal-directed applications we had in mind. Other Forth-based expert system development tools, such as Expert-2 [PAR84], FORPS [MAT87], and FORTES [RED87], offer smaller kernels and higher execution speeds than our Prolog interpreter but suffer from limited usage, nonstandard syntax, and lack of tested problem-solution paradigms.

We feel that our integration of Prolog and Forth has been successful in addressing a number of important design issues. As the controller for the series of vestibular investigations aboard the IML-1 Spacelab mission, our system must observe close tolerances on the timeline and remain well

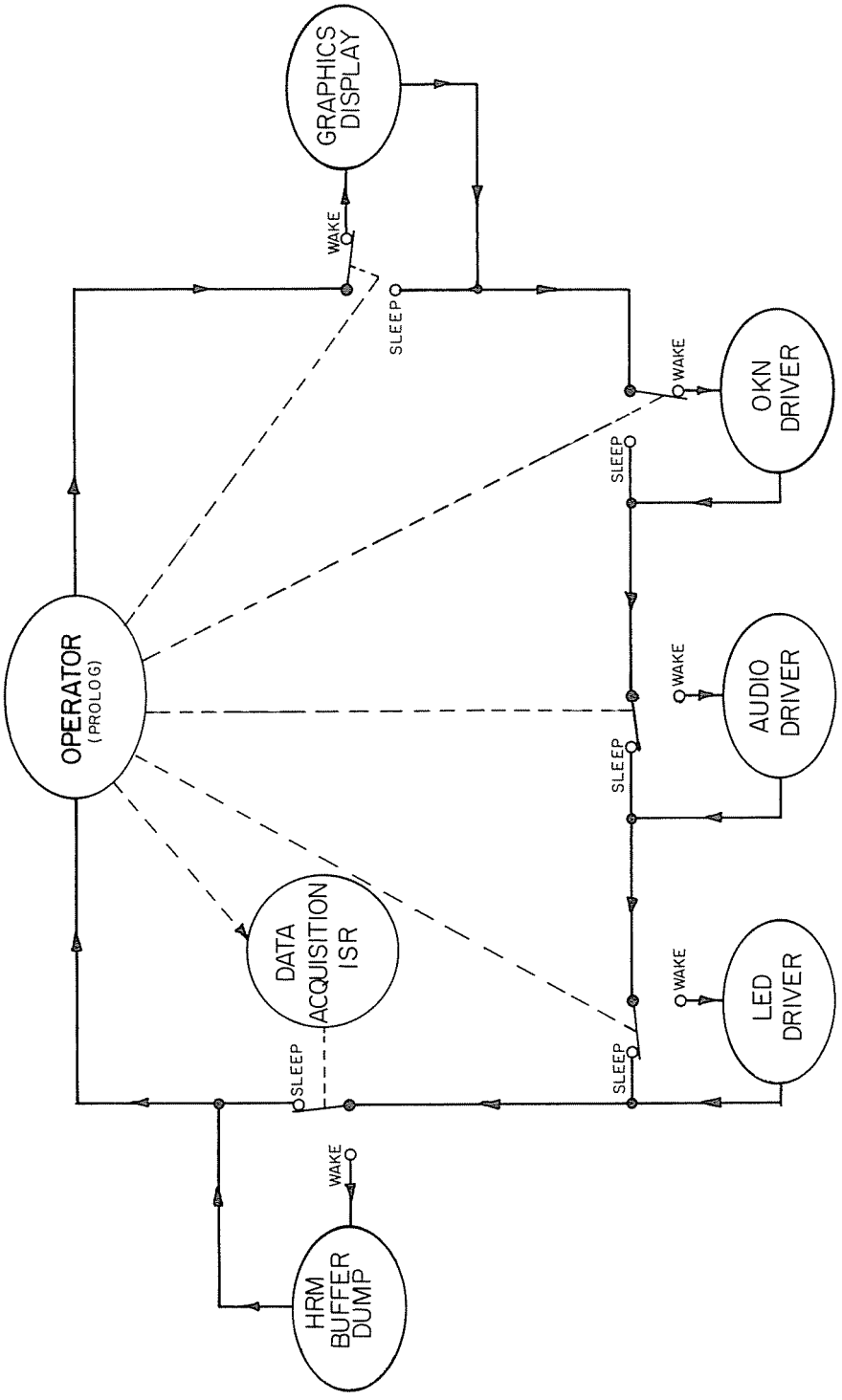

Figure 6
Portion of the MVI expert system round robin multitasking loop.

integrated with the other systems of people and machines on the shuttle. It is therefore imperative that the system provide clear and understandable information to the operator and attempt to reduce operator fatigue and error when on-line. Our approach has been to provide the system with the knowledge it needs to manage these tasks in a variety of situations while simultaneously providing a flexible user interface that allows the operator to exert greater control if conditions warrant. Prolog, being goal-directed and largely declarative, is well suited to this approach and has allowed us to design and implement rapidly.

Another advantage of our Forth-based Prolog is that the data acquisition, control, display, and transmission procedures are readily coded in Forth, permitting us to satisfy speed constraints. By using a multitasking Forth, we were able to implement an architecture that allowed us to avoid having to queue data or delay responding to changes until reasoning stops—in effect permitting asynchronous reasoning where needed. Indeed, the most unique feature of the MVI expert system knowledge base is its procedure base, which comprises all the Prolog objects formed from Forth colon and code definitions using the **builtin** predicate. By constructing this procedure base, all real-time routines could be developed using Forth and yet controlled by the inference mechanism provided by Prolog. This not only separated software development requirements, but also separated execution speed requirements. Consequently, our real-time expert system could be developed for a small computer with a relatively slow version of Prolog.

The procedure base allows the expert system to manage the real-time computing tasks and thereby reduces the complexity of the procedures that the operator needs to perform. Given the suboptimal environment that Spacelab provides for complex scientific experiments and the high cost of losing data from these experiments, this feature is extremely important. The data and rule bases also contribute to reducing the complexity of the operator's task by "knowing" the experimental protocols, equipment setup requirements, and calibration procedures, and by being able to guide the operator through these.

Every real-time computing task does not require an expert system; in fact, few do. As the complexity of the user interface requirements or software logic increases, however, expert system tools should be considered. We feel that our decision to build an expert system to control the MVI experiments reduced the software development effort. Although many of the features provided by the expert system could have been developed using traditional engineering languages such as Forth, Assembler, or Fortran, the initial program development time would have been much longer, and the software modification and maintenance tasks would have become very difficult. Prolog, by its declarative nature, substantially reduced the distance between the programmer (and expert) and the code. This feature allowed rapid prototyping and simplified software maintenance. Modifications to Prolog source code can often be made by life scientists with no computer training!

Successful space-based computing applications often test the engineering skills of their developers. In many cases, a careful consideration of a host of software/hardware performance issues has led to the choice of Forth as the language for development and delivery ([HAR85], [RAS86], [HAR86]). We have shown that the technology exists today to provide viable knowledge system solutions to such applications as an adjunct to the underlying Forth. Furthermore, we believe that our Forth-based Prolog can be used to provide AI solutions to many well-chosen real-time problems.

Acknowledgement

The authors wish to thank Martin Tracy of FORTH, Inc., for his assistance in implementing a polyFORTH version of Prolog, and Carol Verrett for preparing the manuscript. W. H. Paloski, A. J. Krever, and A. K. West were supported by NASA Contract NAS9-17200.

References

[BAR85] Bartoletti, D. C., Lewis, C. S., Paloski, W. H., Odette, L. L. and Yestin, N. S. 1985. Design of a cardiovascular drug knowledge-base for use in critical care monitoring. *Proc. 11th Ann. NorthEast Bioengineering Conference*. Silver Spring, MD: IEEE.

[CLO81] Clocksin, W. F., and Mellish, C. S. 1981. *Programming in Prolog*. Berlin: Springer-Verlag.

[DRE86] Dress, W. B. 1986. REAL-OPS: A real-time engineering applications language for writing expert systems. *J. Forth Appl. and Res.* 4(2):113-24.

[HAR85] Harris, H. M. 1985. Forth as the basis for an integrated operations environment for a space shuttle scientific experiment. *J. Forth Appl. and Res.* 3(2):23-34.

[HAR86] Harris, H. M. The development of an expert system for the command and control of an orbiting spacecraft. *J. Forth Appl. and Res.* 4(2):305.

[JOH83] Johnson, H. E., and Bonissone, P. B. 1983. Expert system for diesel electric locomotive repair. *J. Forth Appl. and Res.* 1(1):7-16.

[LEI87] Leinweber, D. 1987. Expert systems in space. *IEEE Expert* 2(1):26-36.

[MAT86] Matheus, C. J. 1986. The internals of FORPS: A Forth-based production system. *J. Forth Appl. and Res.* 4(1):7-28.

[ODE87] Odette, L. L., and Paloski, W. H. 1987. Use of a Forth-based Prolog for real-time expert systems. II. A full Prolog interpreter embedded in Forth. *J. Forth Appl. and Res.*, this issue.

[PAR84] Park, J. 1984. *Forth expert system*. Mountain View, CA: Mountain View Press.

[PAR86] Park, J. 1986. Expert systems in Forth. *J. Forth Appl. and Res.* 4(1):3-6.

[RAS86] Rash, J. 1986. Prototype expert system in OPS5 for data error detection. *J. Forth Appl. and Res.* 4(2):297-300.

[RED86] Redington, D. 1986. A Forth oriented real-time expert system: A FORTES polysomnographer. *J. Forth Appl. and Res.* 4(1):47-56.

[TRE86] Trelease, R. B. 1986. Implementation of an experimental microcomputer-based medical diagnosis system. *J. Forth Appl. and Res.* 4(1):57-66.

[VAN83] Van DeWalker, R., and Rather, E. D. 1983. *polyFORTH II reference manual*. 4th ed. Hermosa Beach, CA: FORTH, Inc.

[WOL87] Wolfe, A. 1987. An easier way to build a real-time expert system. *Electronics* 60(5):71-3.

Manuscript received August 1986.

Bill Paloski has been building data acquisition and patient monitoring systems since 1977. Before completing his doctorate in biomedical engineering at Rensselaer Polytechnic Institute in 1982, he spent one year at the Oak Ridge National Laboratory and four years at the S. R. Powers Trauma Research Center in Albany, New York. After that he spent three years as an assistant professor at Boston University and then moved to his current position with KRUG International at the Johnson Space Center. In addition to real-time computing, his research interests are in pulmonary physiology, critical care monitoring, and physiological adaptation to weightlessness.

Dr. Odette is international technology marketing manager at Applied Expert Systems, Inc. While at Applied Expert Systems, Dr. Odette has played a major role in designing and implementing the Apex development environment. His work has concentrated on computer language and compiler design. He has also worked on a wide range of technology marketing and delivery issues. Prior to joining Apex, Dr. Odette was a principal of Telphi Systems, Inc., a manufacturer of communications equipment. He was responsible for the design and implementation of data base management systems. Before founding Telphi, Dr. Odette did research at the Massachusetts Institute of Technology where he received a Ph.D. in electrical engineering. He did early work in neural network modeling, where he developed advanced circuit models of neurons. His thesis work focused on perception in the visual system.

Al Krever majored in theater at Emerson College, Boston, Massachusetts, in the mid-1960s but then became interested in special purpose programming. After spending time with DEC and Honeywell in the late 1960s and throughout the 1970s, he joined FORTH, Inc., in 1980. Since then he has been designing custom applications and developing an international reputation as a Forth educator. His current interests are in applying AI to real-time monitoring and process control.

Allison West received a B.S. in biomedical engineering from the University of Iowa in 1981 and an M.S. in electrical science from the University of Michigan in 1984. She currently works for KRUG International, a NASA contractor, in Houston, Texas. Her interests include real-time control, data acquisition systems, and expert systems.