PERSPECTIVES ON THE USE OF RULE-BASED CONTROL

David A. Handelman and Robert F. Stengel
Princeton University
Department of Mechanical & Aerospace Engineering
Princeton, New Jersey

Abstract. This paper addresses issues regarding the application of artificial intelligence techniques to real-time control. Advantages associated with knowledge-based programming are discussed. A proposed rule-based control technique is summarized and applied to the problem of automated aircraft emergency procedure execution. Although emergency procedures are by definition predominately procedural, their numerous evaluation and decision points make a declarative representation of the knowledge they encode highly attractive, resulting in an organized and easily maintained software hierarchy. Simulation results demonstrate that real-time performance can be obtained using a microprocessor-based controller. It is concluded that a rule-based control system design approach may prove more useful than conventional methods under certain circumstances, and that declarative rules with embedded procedural code provide a sound basis for the construction of complex, yet economical, control systems.

Keywords. Aerospace control; artificial intelligence; real-time operation; programming languages; hierarchical systems; man-machine systems; system failure and recovery.

INTRODUCTION

The apparent success of knowledge-based systems, such as expert systems, to provide a limited human-like decision-making capability within a well-defined problem domain gives strong support to their use in the design and implementation of complex control systems [1-4]. Before knowledge-based control techniques are widely accepted, however, many questions regarding their utility must be addressed. Why should a control system designer consider using knowledge-based programming techniques? What distinguishes knowledge-based techniques from conventional ones, how should various knowledge-based control ideas be evaluated and compared, which control problems call for a knowledge-based solution, and what specific benefits result from their use?

This paper attempts to answer some of these questions. The first section provides a comparison between conventional and knowledge-based programming, outlines some advantages associated with knowledge-based systems, and identifies some of the difficulties involved with applying these techniques to time-critical control. The second section summarizes an on-going research effort in real-time knowledge-based control designed to overcome these difficulties. The third section illustrates the utility of the proposed control technique by applying it to the problem of automated aircraft emergency procedure execution, and the final section summarizes benefits associated with its use.

THE PROMISE OF KNOWLEDGE-BASED PROBLEM SOLVING

One way knowledge-based systems can be distinguished from conventional software is by the manner in which data and the routines used to manipulate data remain separated within the program. As opposed to being written in a procedural manner whereby syntactic restrictions dictate an intermixing of code and data, knowledge-based systems use declarative statements, often in the form of rules, to declare and associate pieces of data. As the system runs, modifications and additions to the data are obtained through the use of an inference engine. In this case, the concept of data is generalized to mean knowledge, and the inference engine acts on the knowledge base (previously recognized as the data base) in hopes of inferring additional knowledge from that which already exists. The solution to a problem addressed by a knowledge-based system can come in the form of the additional knowledge (or information) inferred as a result of the system's execution, as well as in the form of sequenced actions performed as side effects of its execution.

The fundamental separation of a knowledge-based system into inference engine and knowledge base results in an enhanced capability for decision making and subsequent problem solving. The value of this enhancement, however, can mean different things to different people. While the user of a knowledge-based system may be impressed by the performance of the program itself, the designer of such a program also will appreciate the convenient environment provided by the adoption of knowledge-based system techniques. For example, although creation of an effective and consistent knowledge base is the toughest part of system construction, the mechanical ease with which it can be prototyped, tested, and modified (given the proper software and hardware tools) reflects a significant increase in programmer productivity in comparison to similar programming efforts based on conventional methods [5,6]. The benefits of using knowledge-based techniques, therefore, include not only what the resulting program can do, but also the efficient manner in which such a program may be created. It is within this context -- increased programmer productivity as well as program performance -- that the utility of knowledge-based techniques should be evaluated.

Many factors complicate the use of knowledge-based systems technology in control. Time-critical and numeric in nature, conventional control algorithms exhibit computational characteristics radically different from those exhibited by knowledge-based systems. The strength of knowledge-based systems comes from a symbolic processing capability, i.e., the ability to reason with non-numeric data. Unfortunately, symbolic computation facilities, in addition to being monetarily expensive, traditionally

Presented at the IFAC Workshop on
Artificial Intelligence in Real-Time Control,
Swansea, UK, September 1988.

result in slow execution speeds (in comparison to numeric facilities) and weak support for the representation and manipulation of floating-point numeric data types. Under the assumption that knowledge-based control efforts should build upon, and not attempt to replace, existing effective numerical control algorithms, a major challenge to control system designers is to integrate efficiently symbolic and numeric computation in a real-time environment.

Most control-oriented real-time knowledge-based systems developed to date can be characterized by (1) a separation within the control system of the symbolic and numeric processing environments (software and/or hardware), and (2) a supervisory role for the knowledge-based system, usually involving monitoring, diagnosis, and planning. Separation of the symbolic and numeric processing environments is justified by the fact that specialized software and hardware exist for this purpose. Moreover, as indicated in Fig. 1, sensing and control functions, conventionally based on numerical algorithms, usually are considered numeric processing tasks, whereas more general control system information processing usually is considered symbolic. Numeric processing historically has demonstrated the capability for high-bandwidth operation, whereas symbolic processing typically has been associated with low-bandwidth operation. Because sensing and control functions impose strict time constraints on control system design, separation of these high-bandwidth operations from the conventionally lower-bandwidth information processing operations seems essential.

Although justified, the separation of symbolic and numeric processing within a control system can limit severely the interaction between processing environments and the throughput of the system as a whole [7,8]. In time-critical applications such as fault-tolerant flight control, the throughput of a knowledge-based control system must be high. The present research goal is to integrate the desirable attributes of procedural and declarative techniques for efficient and effective control system programming, combining convenience in design with speed, economy, and high symbolic/numeric integration in implementation.

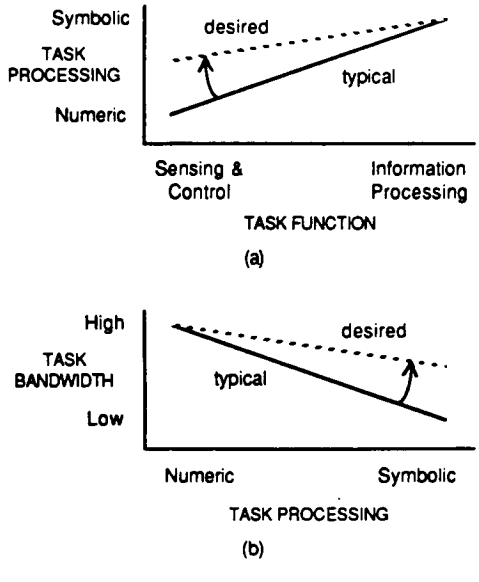


Fig. 1. Characteristic Relationship between Task Function, Processing, and Bandwidth

RULE-BASED CONTROL

Princeton Rule-Based Controller (PRBC) [9-11] embodies a control technique whereby control actions occur as a consequence of search through a knowledge base of parameters, rules, and procedures. Parameters, each with an assigned value, collectively represent a partial description of the "state of the world" pertinent to a given control objective. Search implies the attempt to increase the system's knowledge of the world by inferring additional parameter values, most of which are assumed unknown when the search begins. Information expressing relationships and dependencies between parameters is contained in rules, each of which contains a premise and an action. If a rule premise is true when tested, its action is executed, possibly causing the inference of additional parameter values. Rule testing is guided by an inference engine.

Control actions occur as side effects of search. Buried within the premises and actions of rules are procedures (or calls to procedures) that invoke time-critical control tasks that are best achieved using proven analytical techniques. These procedures are treated as building blocks upon which higher-level control actions are built using rules. Thus, in its simplest form, rule-based search conveniently implements deeply nested IF-THEN-ELSE clauses. The resultant ability to perform complex conditional branching in an organized manner provides a convenient mechanism with which to manipulate analytically derived numerical procedures. In this scenario, it is not the end result of a search that is important to control system operation but the side effects of its execution.

Such use of rules for control results in a tight functional integration between symbolic rules and numerical procedures. In this sense, the technique addresses the desired movement of the curve represented in Fig. 1a by admitting symbolic processing at the sensing and control level. Figure 1b, however, implies a concurrent desire to increase the bandwidth of symbolic processing, thereby addressing the issue of system throughput. The approach adopted here involves two distinct design phases. Phase I involves the development of rules and procedures in separate environments. LISP is used to create and test rules, whereas Pascal is used to derive procedures. Phase II involves final rule and procedure debugging in an integrated Pascal implementation environment. Phase I utilizes a LISP-based inference engine, whereas Phase II uses one based in Pascal. The transition from Phase I to Phase II involves the automatic translation of the LISP-based parameters, rules, and parameter-rule association information into a form acceptable to the Pascal-based inference engine. This knowledge base translation represents a form of automatic code optimization, and results in a compile-ready program implementing a search environment functionally similar to that based in LISP, but exhibiting the proven real-time control system performance characteristics of Pascal [12].

The advantages associated with an integrated Pascal implementation environment are many. In addition to increasing search speed dramatically, knowledge base translation allows rules and procedures to communicate through common data structures. Because parameters are implemented as Pascal variables, their values can be inspected and modified easily by procedures. Similarly, rules are free to access variables other than knowledge base parameters, such as the elements of a matrix routinely used by a numerical procedure. Finally, additional integration results from the ability to place arbitrary Pascal statements within the premises and actions of rules. The search process can thereby invoke timely pieces of Pascal code (calling a procedure, for example) as easily as Pascal code can invoke the process of search.

Although the search technique utilized here for control was inspired by that used in rule-based expert systems, it differs significantly in both implementation and intent. The use of rules in expert systems often is influenced by the characteristics of pure production systems. In production systems, each rule is considered a distinct and modular piece of knowledge, with explicit dependence of one rule on another being discouraged. Rules are meant to communicate with other rules through the indirect and limited link provided by the parameters of the knowledge base, but the intentional calling of one rule by another using parameter values as messages is looked upon with disfavor. As Davis and King [2] point out, "It is the premeditated nature of such message passing (typically in an attempt to 'produce a system with a specified behavior') that is the primary violation of the 'spirit' of production system methodology."

Given that the goal of a control system is, in fact, to produce a system with a specified behavior, it is not surprising that some control system designers view the application of pure production system techniques to control to be misguided. It is the intent of the rule-based control concept discussed here, however, not to take a pure production system approach, but to utilize the declarative power of the production system formalism for its beneficial effects. This effort may be summarized, therefore, as the attainment of procedural activity through the manipulation of declarative expressions, which is not an entirely new idea [13]. The uniqueness in this effort comes from its application to the design and implementation of time-critical control systems, particularly flight control systems. Although violating the spirit of pure production systems in order to provide the specified behavior demanded for control, the proposed control technique retains many advantages related to production systems. Davis and King [2] provide an interesting perspective related to program performance and programmer productivity as mentioned above.

> Since it is possible to imagine coding any given Turing machine in either procedural or production system terms, in the formal sense their computational power is equivalent. This suggests that, given sufficient effort, they are ultimately capable of solving the same problems. The issues we wish to examine are not, however, questions of absolute computational power but the impact of a particular methodology on program structure, as well as of the relative ease or difficulty with which certain capabilities can be achieved.

The next section illustrates the straightforward manner in which significant control system capabilities may be obtained using rules.

DECLARATIVE EXECUTION OF AIRCRAFT EMERGENCY PROCEDURES

The rule-based control technique described above was developed as part of a research program in applications of artificial intelligence theory to fault-tolerant flight control [14]. Presently, the technique is being applied to the automation of aircraft emergency procedures. Aircraft emergencies require a quick response and relatively complex decision making by a flight crew. Prescribed emergency procedures contained in the Pilot's Operating Handbook are designed to guide the flight crew through the decision-making process. Although these emergency procedures are by definition predominately procedural, their numerous evaluation and decision points make a declarative representation of the knowledge they encode highly attractive.

Consider as a simple example the following excerpt of an emergency procedure associated with the electrical system of the CH-47C tandem-rotor helicopter [15]:

> 4-96. FAILURE OF ONE AC GENERATOR.
>
> 4-97. Should one ac generator fail, the remaining ac generator will assume the entire load under normal conditions. This condition will be noticed by the illumination of a generator caution light and by a zero indication on the ac loadmeter for that generator. Attempt to place the inoperative ac generator into operation by performing the following:
>
> a. Master caution lights - PUSH TO RESET.
>
> b. All circuit breakers - CHECK.
>
> c. Generator switch - TEST, then RESET, then ON. If the generator is inoperative, move the generator switch to OFF. (Move the generator switch to TEST and observe the generator caution light. If the caution light goes out, the generator is delivering proper voltage and frequency and a short circuit on a bus is indicated. If the caution light remains on, the generator is inoperative.)

The following discussion highlights how this emergency procedure (with reference to Generator No.1 only) can be made to execute automatically as a side effect of search through a knowledge base of parameters and rules. The inference engine executes on the highest level a repetitive cycle involving knowledge base initialization and goal-directed search on a parameter named PROC_SEARCH_COMPLETED. This is represented in Pascal as

```
repeat
   initialize_knowledge_base;
   determine_value_of(PROC_SEARCH_COMPLETED)
until false;
```

Knowledge base initialization assigns the value of UNKNOWN to all parameters without a stored "initial value". Goal-directed (backward-chaining) search on parameter PROC_SEARCH_COMPLETED results in the inference engine testing rules capable of supplying a value for this parameter in their action, such as the following.

```
[RULE_PROC_1
  [PREMISE
    '($OR ($EQ PROC_REQUIRED      'FALSE)
          ($EQ PROC_STEP_CHECKED 'TRUE]
  [ACTION
    '($SETQ PROC_SEARCH_COMPLETED 'TRUE]]
```

Note that the premise of this rule depends on other parameters whose values also are as yet unknown (parameters receiving a value of UNKNOWN during knowledge base initialization are written entirely in capital letters). When tested, the premise first searches for a value for parameter PROC_REQUIRED. A value of FALSE implies that the emergency procedure is not required (the conditions under which it is to be executed do not exist). The following rules determine whether or not procedure requirements are met.

```
[RULE_PROC_REQ_1
  [PREMISE
   '($AND ($EQ Proc_Step 0)
          ($EQ Gen_1_Control_Switch_Status 'ON)
          ($EQ Gen_1_Loadmeter_Status     0)
          ($EQ Gen_1_Caution_Light_Status 'ON)
          ($EQ Gen_2_Caution_Light_Status 'OFF]
  [ACTION
    '(($SETQ Proc_Step 1)
      ($SETQ PROC_REQUIRED 'TRUE]]
[RULE_PROC_REQ_2
  [PREMISE
    '($GT Proc_Step 0)
  [ACTION
    '($SETQ PROC_REQUIRED 'TRUE]]

[RULE_PROC_REQ_3
  [PREMISE
    '($EQ Rule_Tested 'TRUE]
  [ACTION
    '($SETQ PROC_REQUIRED 'FALSE]]
```

Rules are tested in order of appearance in the knowledge base until a value for the specified parameter is obtained. These three rules are tested in order when a value for PROC_REQUIRED is needed within the premise of RULE_PROC_1. The premise of the first rule holds if the procedure is not already executing (the active procedure step number, represented by parameter Proc_Step, is 0) and the conditions specified by the remainder of the $AND clause prevail. The second rule holds if the procedure is already executing (Proc_Step is greater than 0), and the third rule always holds whenever tested. Thus, if during search only the third rule holds, then PROC_REQUIRED obtains a value of FALSE, and PROC_SEARCH_COMPLETED is assigned a value of TRUE within the action of RULE_PROC_1 quickly ending the search cycle. However, if RULE_PROC_REQ_1 holds, then the step number is set to 1, PROC_REQUIRED obtains a value of TRUE, the first subclause of RULE_PROC_1 fails, and a goal-directed search for PROC_STEP_CHECKED begins. A similar search will prevail if RULE_PROC_REQ_1 fails, but RULE_PROC_REQ_2 holds.

Emergency procedures are executed here as a series of steps. Steps are executed one at a time, in order. Each step in general involves an initial action (invoked for Step 1 by parameter PROC_STEP_1_STARTED), a time delay (reflected by parameter Proc_Step_Delay) over which the initial action is allowed to take effect, and a final action (invoked for Step 1 by parameter PROC_STEP_1_FINISHED). For example, an emergency procedure may require the flipping of a switch and the monitoring of its effect. The time delay between the initial and final step actions gives the physical system being probed a chance to react. When a final step action is taken, Proc_Step is incremented, allowing the next emergency procedure step to be performed.

Rules invoked by cyclic goal-directed search effectively encode the logic required to execute these emergency procedure steps with the appropriate time delays. As shown by the rules above, cyclic search on parameter PROC_SEARCH_COMPLETED results in cyclic search on parameter PROC_STEP_CHECKED (assuming that PROC_REQUIRED is TRUE). In effect, the control system continually asks the question, "Has the appropriate emergency procedure step been checked?" Rules capable of answering this question for Step 1 are shown below.

```
[RULE_PROC_STEP_1A
  [PREMISE
    '($AND ($EQ Proc_Step 1)
           ($EQ Proc_Step_Timer     'STOPPED)
           ($EQ PROC_STEP_TIMER_SET 'TRUE)
           ($EQ PROC_STEP_1_STARTED 'TRUE]]
  [ACTION
    '($SETQ PROC_STEP_CHECKED 'TRUE]]

[RULE_PROC_STEP_1B
  [PREMISE
    '($AND
      ($EQ Proc_Step 1)
      ($EQ Proc_Step_Timer 'RUNNING)
      ($OR [$AND
             ($EQ PROC_STEP_1_BYPASSED 'TRUE)
             ($EQ PROC_STEP_TIMER_SET  'TRUE]
           ($EQ PROC_STEP_TIMEOUT 'FALSE]]
  [ACTION
    '($SETQ PROC_STEP_CHECKED 'TRUE]]

[RULE_PROC_STEP_1C
  [PREMISE
    '($AND ($EQ Proc_Step 1)
           ($EQ Proc_Step_Timer      'RUNNING)
           ($EQ PROC_STEP_TIMEOUT    'TRUE)
           ($EQ PROC_STEP_1_FINISHED 'TRUE)
           ($EQ PROC_STEP_TIMER_SET  'TRUE]]
  [ACTION
    '($SETQ PROC_STEP_CHECKED 'TRUE]]
```

When a value for PROC_STEP_CHECKED is required by RULE_PROC_1, these rules are tested. Each rule performs a distinct intra-step function, monitoring the "state" of Procedure Step 1 and quickly performing a piece of the step if required. At least one rule always holds. Within the first rule, the parameter Proc_Step_Timer is checked for a value of STOPPED. This indicates that the step has not begun, in which case the timer is started (via search for parameter PROC_STEP_TIMER_SET using rules not shown) and the initial step action is taken (via search for parameter PROC_STEP_1_STARTED). If the first rule fails (Proc_Step_Timer is RUNNING), a search for parameter PROC_STEP_1_BYPASSED within the second rule checks whether or not the final step action already has been performed by the flight crew. If it has, PROC_STEP_1_BYPASSED will return a value of TRUE and a search on PROC_STEP_TIMER_SET will stop the timer and increment Proc_Step. If PROC_STEP_1_BYPASSED is FALSE, the next premise subclause tests whether or not the required step time delay has been achieved. If PROC_STEP_TIMEOUT is FALSE (inferred with a rule not shown), the rule premise holds without performing any emergency procedure actions. If PROC_STEP_TIMEOUT is TRUE, the third rule performs the final step action (via search for parameter PROC_STEP_1_FINISHED) and stops the timer and increments Proc_Step (via search for parameter PROC_STEP_TIMER_SET).

Thus, each search for parameter PROC_STEP_CHECKED performs a single piece of a single procedure step. Execution of the entire emergency procedure requires many search cycles. The following three Step 1 rules encode the first part of the emergency procedure excerpt shown above.

```
[RULE_PROC_STEP_1_ST
  [PREMISE
    '($AND ($EQ Advisory_Mode 'ON)
           ($PASCAL "advise
             ('*** EMERGENCY PROCEDURE ***
               Failure of One AC Generator:
               a. Master caution light -
                  PUSH TO RESET');")
           ($SETQ Proc_Step_Delay 1.0]
  [ACTION
    '($SETQ PROC_STEP_1_STARTED 'TRUE]]

[RULE_PROC_STEP_1_BY
  [PREMISE
    '($EQ Master_Caution_Light_Status 'OFF]
  [ACTION
    '($SETQ PROC_STEP_1_BYPASSED 'TRUE]]

[RULE_PROC_STEP_1_FIN
  [PREMISE
   '($AND
     ($EQ   Operational_Mode           'ACTIVE)
     ($SETQ Master_Caution_Light_Command 'PUSH]
  [ACTION
    '($SETQ PROC_STEP_1_FINISHED 'TRUE]]
```

The premise of the first rule contains three subclauses. The first subclause holds if parameter Advisory_Mode has value ON. If so, the next two subclauses are evaluated. The $PASCAL subclause contains embedded Pascal code that calls a procedure which sends an advisory message to the cockpit for evaluation by the flight crew. The $SETQ subclause assigns to parameter Proc_Step_Delay a value of 1. Both of these subclause operators, as members of a Boolean premise, always return TRUE. Similarly, the premise of the third rule generates a master caution light command within the $SETQ subclause if parameter Operational_Mode has value ACTIVE. Thus, for Procedure Step 1, if Advisory_Mode is ON and Operational_Mode is ACTIVE, the control system will reset the master caution light if after 1 sec this task has not been performed by the flight crew. In general, the parameter Advisory_Mode can have values ON and OFF, and parameter Operational_Mode can have values ACTIVE and STANDBY. If Advisory_Mode is OFF, no recommendations are sent to the flight crew, and advisory delays are eliminated. If Operational_Mode is STANDBY, actions may be recommended but not executed by the flight control system.

If any of these three rules fails, an additional rule supplying an appropriate "default" value for the required parameter is tested, such as the following.

```
[RULE_PROC_STEP_1_BY_0
  [PREMISE
    '($EQ Rule_Tested 'TRUE]
  [ACTION
    '($SETQ PROC_STEP_1_BYPASSED 'FALSE]]
```

The remaining parts of the emergency procedure excerpt given above are implemented in a similar fashion. Consider the next three rules.

```
[RULE_PROC_STEP_2_ST
  [PREMISE
    '($AND
       ($EQ Advisory_Mode 'ON)
       ($PASCAL "advise
         ('b. All circuit breakers - CHECK');")
       ($SETQ Proc_Step_Delay 2.0]
  [ACTION
    '($SETQ PROC_STEP_2_STARTED 'TRUE]]

[RULE_PROC_STEP_2_BY
  [PREMISE
    '($EQ BREAKERS_CHECKED_BY_FLIGHTCREW 'TRUE]
  [ACTION
    '($SETQ PROC_STEP_2_BYPASSED 'TRUE]]

[RULE_PROC_STEP_2_FIN
  [PREMISE
    '($AND ($EQ OPERATIONAL_MODE      'ACTIVE)
           ($EQ BREAKERS_CHECKED_BY_FCS 'TRUE]
  [ACTION
    '($SETQ PROC_STEP_2_FINISHED 'TRUE]]
```

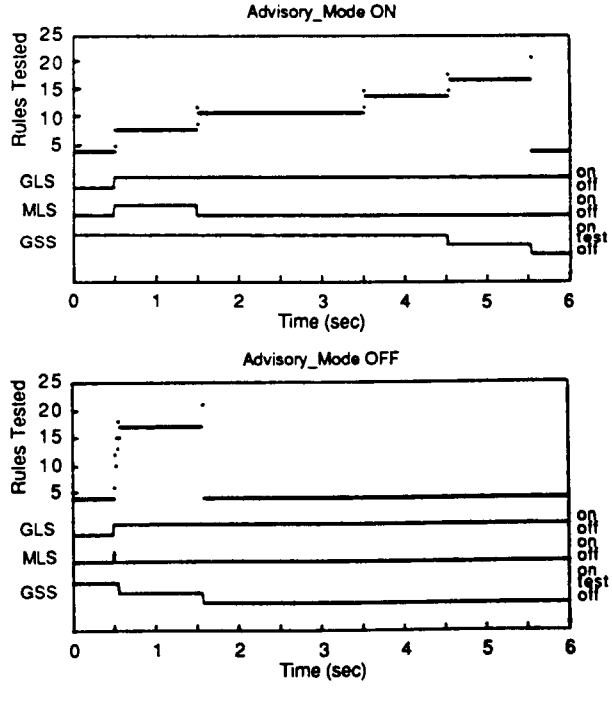
The first rule provides an advisory with a 2 sec grace period. Within this time period, a goal-directed search on parameter BREAKERS_CHECKED_BY_FLIGHTCREW performed within the second rule invokes additional rules that check if all circuit breakers have been checked by the flight crew. The third rule invokes other rules performing this task automatically if and when necessary.

The last part of the emergency procedure is implemented with two steps. Rules of Procedure Step 3 verify movement of the generator switch to the TEST position. If Advisory_Mode is OFF, the control system performs the action immediately. If Advisory_Mode is ON, a maximum advisory delay of 1 sec is tolerated. Finally, rules of Procedure Step 4 wait one more second (regardless of Advisory_Mode) before moving the generator switch to ON or OFF, depending on the status of the generator caution light.

The result of using a declarative rule-based representation for emergency procedure execution is an organized and easily maintained software hierarchy. Additionally, by using cyclic search with "time-sliced" procedure steps, the action of the control system can be made to emulate a multi-tasking operating system. This capability is demonstrated below.

Figure 2 shows a time history of the amount of "effort" expended by a single-processor rule-based controller executing the emergency procedure described above. The data was obtained with the Princeton Rule-Based Controller Development System [9], employing a commercially-available single-board computer outfitted with an 8-MHz 80286 processor and an 8-MHz 80287 math coprocessor. The controller knowledge base contained 34 parameters and 34 rules. Off-line LISP-to-Pascal knowledge base translation required 8.6 min on a personal computer functionally similar to the PRBC processor described above. Pascal representation of the knowledge base required 16 KBytes of Random-Access Memory (some as code, some as data), and the inference engine required 13 KBytes of code.

During simulation runs, certain parameters were given a fixed initial value: Gen_1_Loadmeter_Status was 0, Gen_2_Caution_Light_Status was OFF, BREAKERS_CHECKED_BY_FLIGHTCREW was FALSE, BREAKERS_CHECKED_BY_FCS was TRUE. Emergency procedure execution was triggered by changing the value of Gen_1_Caution_Light_Status from OFF to ON. Each data point in the top plots of Fig. 2 corresponds to the number of rules tested by the inference engine during a goal-directed search cycle. Adjacent data points are separated by the amount of time required to complete a search cycle. Figure 2 shows that although the value of the parameter Advisory_Mode has a large effect on emergency procedure step timing (as intended), the inference engine consistently executes search cycles at a very high rate.



GLS = Gen_1_Caution_Light_Status
MLS = Master_Caution_Light_Status
GSS = Gen_1_Control_Switch_Status

Fig. 2. Execution of Single Emergency Procedure

Figure 3 shows a time history of rule testings performed by the same rule-based controller concurrently executing five copies of the same emergency procedure. Although all procedures trigger off the same generator caution light, each manipulates its own master caution light and generator switch, and employs a unique set of step delays. Figure 3 shows that this version of the rule-based controller, with a knowledge base of 154 parameters and 170 rules, still provides real-time multi-tasking performance with a single economical processor.

CONCLUSIONS

The main conclusion to be drawn is that a rule-based control design approach may prove more useful than conventional methods under certain circumstances, especially when complex decision making is required. The proposed rule-based control technique provides basic programming constructs required in real-time applications such as flight control. Capabilities including event scheduling, selection, and synchronization, as well as data passing and sharing, are implemented in an extremely flexible and modular representation. Consequently, declarative rules with embedded procedural code provide a sound basis for the construction of complex, yet economical, control systems.

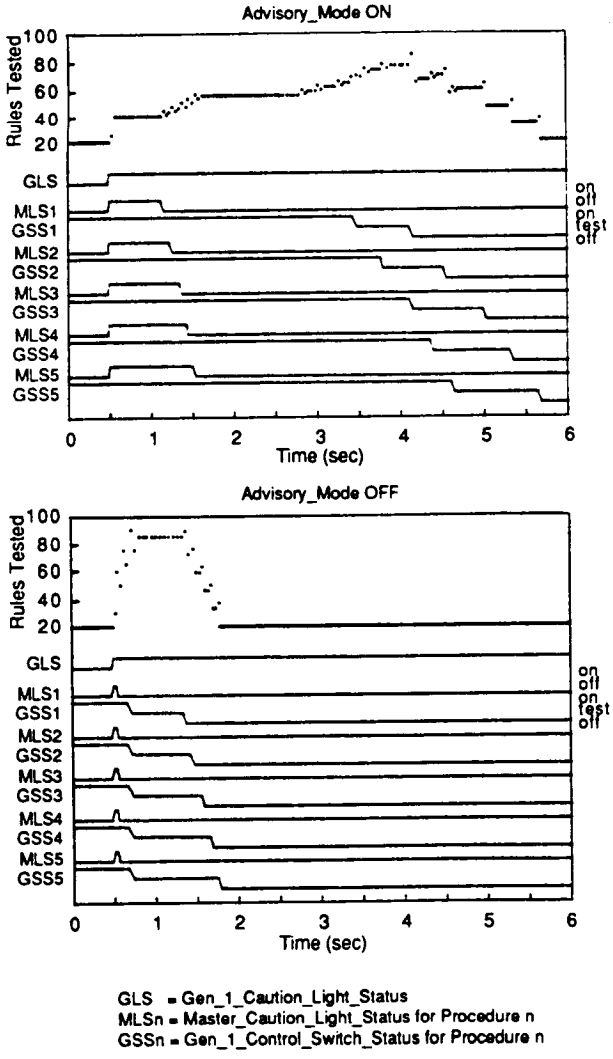


Fig. 3. Execution of Multiple Concurrent Emergency Procedures

ACKNOWLEDGMENT

This project was sponsored by the U. S. Army Research Office under Contract No. DAAG29-84-K-0048.

REFERENCES

1. Barr, A., Cohen, P., and Feigenbaum, E., The Handbook of Artificial Intelligence, William Kaufmann, Los Altos, California, 1982.

2. Davis, R. and King, J. J. in Buchanan, B., and Shortliffe, E., Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley Publishing Company, Reading, Mass., 1984.

3. Hankins, W., Pennington, J., and Barker, L., "Decision-Making and Problem-Solving Methods in Automation Technology," NASA-TM-83216, May 1983.

4. James, J., "A Survey of Knowledge-Based Systems for Computer-Aided Control System Design," Proc. 1987 American Control Conference, Minneapolis, June 1987.

5. Ricks, W., and Abbott, K., "Traditional Versus Rule-Based Programming Techniques: Application to the Control of Optional Flight Information," Applications of Artificial Intelligence V, Proc. SPIE, Vol. 786, May 1987.

6. Hueschen, R., and McManus, J., "Application of AI Methods to Aircraft Guidance and Control," Proc. 1988 American Control Conference, Atlanta, June 1988.

7. O'Reilly, C., and Cromarty, A., "'Fast' is not 'Real-Time': Designing Effective Real-Time AI Systems," Applications of Artificial Intelligence II, Proc. SPIE 548, 1985.

8. Birdwell, J., Cockett, J., and Gabriel, J., "Domains of Artificial Intelligence Relevant to Systems," Proc. 1986 American Control Conference, Seattle, June 1986.

9. Handelman, D., and Stengel, R., "An Architecture for Real-Time Rule-Based Control," Proc. 1987 American Control Conference, Minneapolis, June 1987.

10. Handelman, D., and Stengel, R., "Combining Expert System and Analytical Redundancy Concepts for Fault-Tolerant Flight Control," AIAA J. Guidance, Control, and Dynamics, Vol. 11, Jan-Feb, 1989.

11. Handelman, D., and Stengel, R., "Rule-Based Mechanisms of Learning for Intelligent Adaptive Flight Control," Proc. 1988 American Control Conference, Atlanta, June 1988.

12. Westermeier, T., and Hansen, H., "The Use of High Order Languages in Microprocessor-Based Systems," Proc. 1984 American Control Conference, San Diego, June 1984.

13. Munakata, T., "Procedurally Oriented Programming Techniques in Prolog," IEEE Expert, Vol. 1, No. 2, Summer 1986.

14. Stengel, R., "Artificial Intelligence Theory and Reconfigurable Control Systems," Department of Mechanical and Aerospace Engineering Report 1664, Princeton University, Princeton, New Jersey, June 1984.

15. "Operator's Manual: Army Model CH-47B and CH-47C Helicopters," Army Technical Manual No. 55-1520-227-10, Aug. 1970.