Expert System for Real-Time Aircraft Monitoring

Joey B. Flanders*
Air Force Flight Test Center, Edwards Air Force Base, California 93523
Charles H. Jones†
Washington State University, Pullman, Washington 99164
and
Robin M. Madison‡
Air Force Flight Test Center, Edwards Air Force Base, California 93523

Real-time aircraft testing is increasing in both frequency and complexity. Greater demands are being placed on flight test engineers to simultaneously monitor aircraft systems during flight and to perform problem analysis real time. This has caused a rise in requirements for automated and intelligent monitoring systems. The use of state-of-the-art off-the-shelf hardware and software has provided a cost effective base for supporting development of real-time expert systems to aid in these monitoring tasks. A brief introduction to the telemetry environment is given. This is followed by descriptions of some specific applications including a propulsion monitor. The final section discusses some aspects of knowledge engineering and an inherent conflict between knowledge engineering and software engineering.

Introduction

DURING the 1980s, the NASA Johnson Space Center (JSC) recognized a need for a technology upgrade for their support of Space Shuttle missions. At the same time, advances in the area of artificial intelligence known as expert systems were significant and JSC had recently developed the C Language Integration Production System (CLIPS) as a readily available forward chaining inference engine. This all led to the work described by Heindel et al.[1] and Muratore.[2]

In 1988, Edwards Air Force Base instigated a project to transfer this technology and implement a similar system for aircraft mission support. The F15 short takeoff and landing (STOL) maneuver technology demonstrator became the initial test bed, and by the middle of 1989, a working prototype had been developed at Edwards. Personnel at Edwards then became aware of the work by Disbrow et al.[3] and a decision was quickly made to transport, enhance, and mold that program to requirements at Edwards starting with the STOL.

The software development presented in this paper used the hardware configuration, data acquisition software, and general system design from JSC. Adding to this the work of Disbrow et al.,[3] technology transfers provided a solid and necessary base for the higher level work of expert system development.

Design

The general system design is a three-layered architecture where each layer is dependent on the one below it. For flexibility, the flight test engineer has visibility into all three layers (see Fig. 1).

Layer 1 provides telemetry data acquisitioning. This layer is the lifeblood of the telemetry monitoring system since it gets the data (real time) from the aircraft into the computer system. Several functions are accomplished such as bit and frame synchronization, extraction and decommutation of parameters from the telemetry stream, and primitive calibration of parameters.

Layer 2 provides information gathering. This is where the raw data from layer 1 are first massaged. This includes parameter calibration, unit conversion, filtering, and other system specific algorithms. It also includes intelligent graphical display of these massaged data. Examples of applications at this layer are the discrete monitor and the graphical displays of the nozzle schematic and propulsion expert system described later.

Layer 3 provides knowledge application. This consists of assessments that are made by knowledge-based expert systems with enhanced man-machine interfaces. Heuristic inferencing is accomplished through LISP-like sets of rules used by CLIPS. Two applications that have been developed at this layer are the nozzle schematic and propulsion expert system described later.

Here is a conceptual example of the three layers. At layer 1, there are raw data: "The engine temperature is 500 deg." At layer 2, there is an interpretation of a sequence of raw data points. This might provide "The engine temperature is rising" and a display might turn red if the temperature reaches some threshold. At layer 3, sets of information are synthesized, knowledge is applied to them, and conclusions are drawn: "If the temperature continues to rise, damage will occur, therefore, turn the engine off."

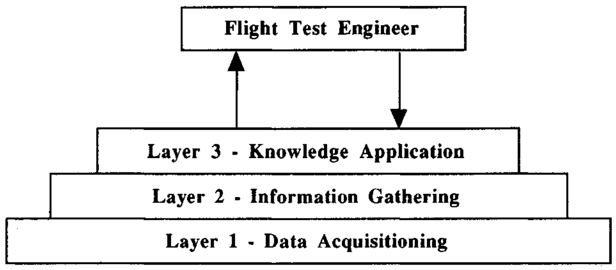

Flight Test Engineer
Layer 3 - Knowledge Application
Layer 2 - Information Gathering
Layer 1 - Data Acquisitioning

Fig. 1 Three-layer design.

Presented as Paper 90-1311 at the AIAA/SFTE/DGLR/SETP Fifth Biannual Flight Test Conference, Ontario, CA, May 21–24, 1990; received April 23, 1990; revision received Dec. 5, 1990; accepted for publication Jan. 16, 1991. This paper is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

*Mathematician.

†Graduate Student.

‡Supervisory Mathematician.

Telemetry Data Acquisition

Acquisition of pulse code modulation (PCM) telemetry data is accomplished through several steps (see Fig. 2). The telemetry signal is generated on the aircraft, received by antenna, and transmitted to a decommutation system. After decommutation, the data is transferred to shared memory in a workstation via direct memory access (DMA). This layer 1 process has critical speed and buffering requirements. The existence of these commercial off-the-shelf systems (which meet these requirements and provide complete layer 1 support) is crucial to a timely, cost-effective development.

Speed requirements are met in essentially two ways. First, the DMA allows the data to be directly input into the workstation's main memory from the decommutation system. Second, a shared memory architecture allows several applications to access the data from a single location in main memory.

The buffering is accomplished by a software architecture imposed on shared memory. The data comes across the DMA and is put into a ring buffer. From the ring buffer, data is transferred to one of a series of format independent storage arrays (FISA). The applications then obtain data from the most recently filled FISA. The ring buffer allows for an uninterrupted data flow from the decommutation system, whereas the FISAs allow for buffering into the application programs.

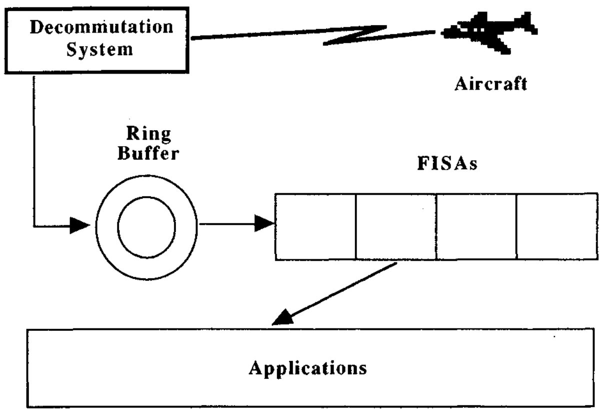


Fig. 2 Telemetry data acquisition.

Ideally, the buffering should prevent loss of data. In practice, though, the data rates (as much as 100 or 1000 samples/s) are considerably higher than the screen update rates (usually 5–20 updates/s). Thus, the buffering acts as a method for accessing the most current information without interrupting the data flow.

Application Programs

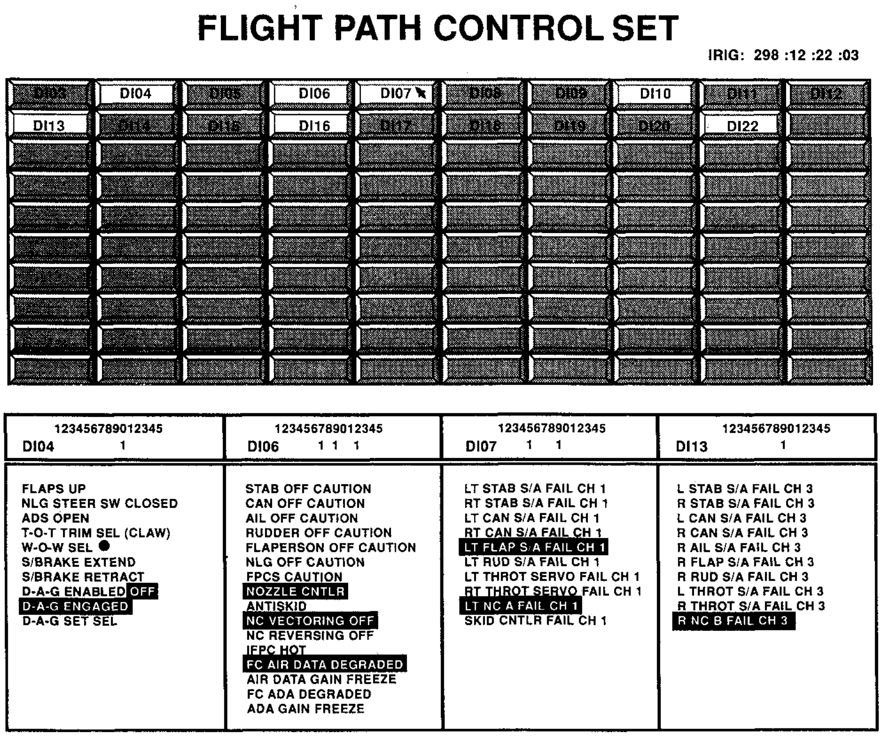
To appreciate the technical advance represented by the following displays, it is helpful to compare them with conventional aircraft monitoring systems. Conventional systems have been oriented around strictly numerical and bit pattern displays. Thus, many monitoring screens are simply columns of numbers with little indication of what these numbers mean. First steps toward providing intelligent monitoring are providing graphical displays, organizing the displayed data conceptually, and providing more complete verbal descriptions of displayed data. Once the man-machine interface is thus enhanced, rule-based inferencing can be added to drive displays.

Discrete Monitor

Over the last decade, aircraft instrumentation has turned more and more toward discrete (on/off) indicators. These discretes can number in the hundreds or even thousands. The conventional approach of simply displaying bit patterns requires an expert and/or paper document lookup to interpret the patterns. This requires a level of concentration and attention that is virtually impossible to maintain for extended periods of time. Figure 3 shows a discrete monitor that eliminates these requirements by providing color indication of status along with immediately available descriptions.

The upper region of the display has several buttons that represent the parent discretes (e.g., DI03–DI22). Each parent discrete consists of 16-bit integer words. A child discrete is made up of one or more bits within a respective parent discrete. Whenever an individual child discrete gets set (i.e., = 1), the button for the corresponding parent discrete turns red. Otherwise, the buttons remain blue.

In the lower region of the display, boxes of detailed information on four respective parent discretes are presented. Detailed information about a particular parent discrete can be



123456789012345 DI04 1	123456789012345 DI06 1 1 1	123456789012345 DI07 1 1	123456789012345 DI13 1
FLAPS UP NLG STEER SW CLOSED ADS OPEN T-O-T TRIM SEL (CLAW) W-O-W SEL ● S/BRAKE EXTEND S/BRAKE RETRACT D-A-G ENABLED OFF D-A-G ENGAGED D-A-G SET SEL	STAB OFF CAUTION CAN OFF CAUTION AIL OFF CAUTION RUDDER OFF CAUTION FLAPERSON OFF CAUTION NLG OFF CAUTION FPCS CAUTION NOZZLE CNTLR ANTISKID NC VECTORING OFF NC REVERSING OFF IFPC HOT FC AIR DATA DEGRADED AIR DATA GAIN FREEZE FC ADA DEGRADED ADA GAIN FREEZE	LT STAB S/A FAIL CH 1 RT STAB S/A FAIL CH 1 LT CAN S/A FAIL CH 1 RT CAN S/A FAIL CH 1 LT FLAP S/A FAIL CH 1 LT RUD S/A FAIL CH 1 LT THROT SERVO FAIL CH 1 RT THROT SERVO FAIL CH 1 LT NC A FAIL CH 1 SKID CNTLR FAIL CH 1	L STAB S/A FAIL CH 3 R STAB S/A FAIL CH 3 L CAN S/A FAIL CH 3 R CAN S/A FAIL CH 3 R AIL S/A FAIL CH 3 R FLAP S/A FAIL CH 3 R RUD S/A FAIL CH 3 L THROT S/A FAIL CH 3 R THROT S/A FAIL CH 3 R NC B FAIL CH 3

Fig. 3 Discrete monitor display.

displayed in a box by choosing the respective parent button with the mouse. Each box contains the parent discrete name, bit pattern information (indicating 1 for the corresponding child bit(s) triggered), respective child messages, and various yellow status descriptions. A red child message indicates that its respective bit(s) has been set. Otherwise, the child message or description remains green.

Nozzle Schematic

The nozzle schematic display shown in Fig. 4 is an application developed specifically for the STOL project. However, the concept behind the display is generic. Wiring schematics play an important role in engineering. They provide a cognitive model and a point of communication. By implementing this cognitive model on a display, the cognitive load of the person monitoring is decreased and nonexperts can understand displayed data rapidly and more easily.

In order to perform the STOL maneuvers, the aircraft has nozzles on the engines that provide thrust vectoring and reversing. Each nozzle has five hydraulic actuators to provide the vectoring motion. The screen display illustrates both left and right engines each with nozzle controller (computer) and hydraulic actuators. Within each nozzle controller, there are two channels (A and B) that provide redundant control of the actuators. Both channels are capable of fully controlling all five actuators. The wiring schematic between the controllers and the actuators represent the state of this control. Each of the various boxes contain redundant pairs of monitoring discretes. The five boxes respective to each engine contain discretes that indicate actuator failures. The discretes within the controller box represent communication between the channels and a higher level of failure modes.

The state of the actuators and control are indicated by changing colors of the appropriate icon. Both the discretes and the control have three states. In the normal mode of operation, all icons are light blue. This indicates that no failures have been detected and that both channels are providing 50% of the current necessary to operate the actuators. When a failure occurs, or a channel stops controlling an actuator, the appropriate icon turns red. The corresponding redundant icon turns green indicating that monitoring or controlling is now totally dependent on the nonfailed channel. When an actuator indicates a failure to both channels, a depower command is issued and a fail-safe message is posted.

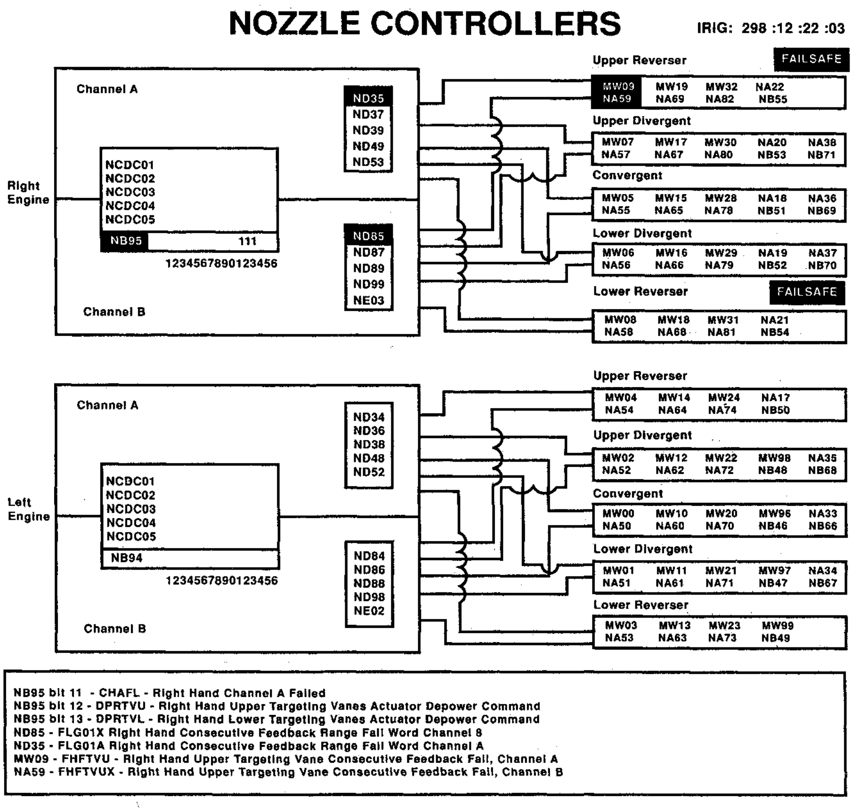
CLIPS is used to control the states of the icons. However, the amount of inferencing here is minimal. This display represents an initial step from layer 2 to layer 3. The following display is a significantly more sophisticated layer 3 application.

Propulsion Expert System

Modern propulsion systems are distinctly complex and monitoring them with conventional systems requires a high level of expertise. Figure 5 displays the engines and associated data in a manner relatively easy to understand. There are four main areas of the display: general aircraft information and data acquisitioning status, strip charts, the expert system message displays, and the engine diagrams.

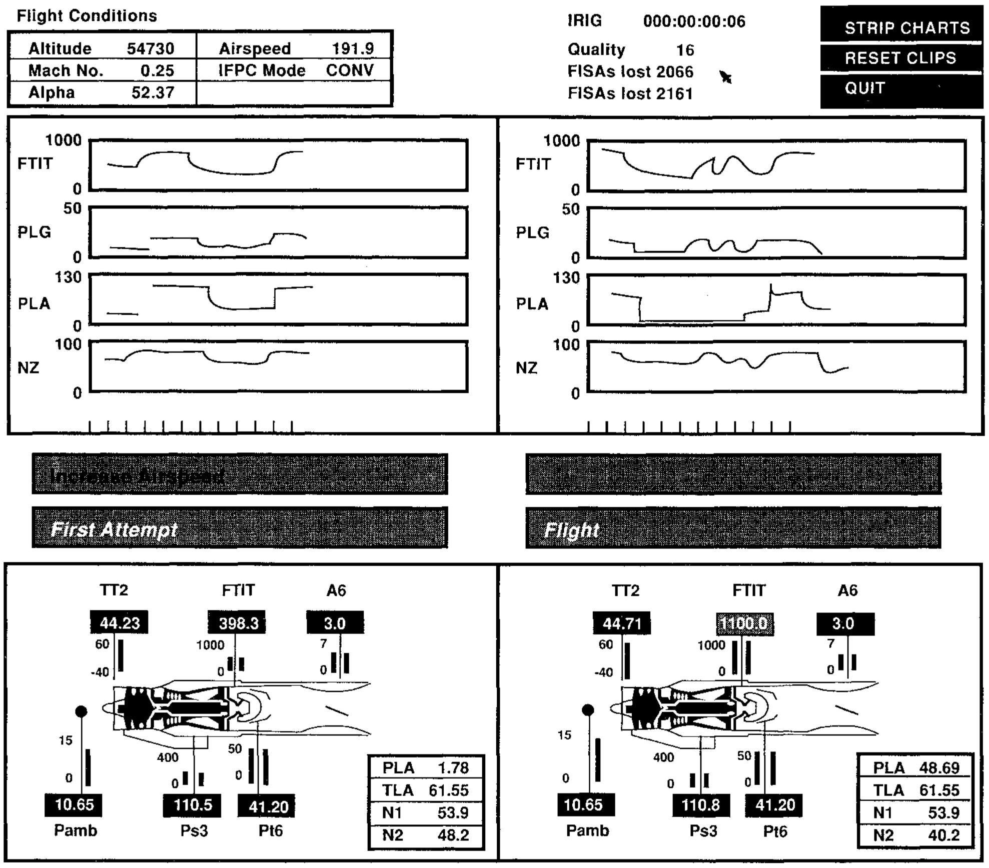
The strip charts display user selected parameters. Surprisingly, although the strip charts are useful for this display, flight test engineers have not been enthusiastic about cathode ray tube (CRT) driven strip charts. This is primarily due to the desire for hard copies that can be taken with them directly after a mission. Hard copies in their hands provide a means for immediate analysis without the cost of system facilities to do a playback.

The thermometer-like gauges above the engine diagram show different temperatures and the ones below the diagram show different pressures. The gauges are placed so that the parameters displayed are in the appropriate position relative to the actual engine. The bars on the right side of the gauges indicate actual values, whereas the bars on the left side indicate a predicted value to indicate a trend in the data. The STOL's main interest is in the vectoring nozzles. Thus, the end of the engine has several different configurations and red arrows indicate the vectoring angle of the nozzles and reverser vanes.



```
NB95 bit 11  - CHAFL - Right Hand Channel A Failed
NB95 bit 12 - DPRTVU - Right Hand Upper Targeting Vanes Actuator Depower Command
NB95 bit 13 - DPRTVL - Right Hand Lower Targeting Vanes Actuator Depower Command
ND85 - FLG01X Right Hand Consecutive Feedback Range Fail Word Channel 8
ND35 - FLG01A Right Hand Consecutive Feedback Range Fail Word Channel A
MW09 - FHFTVU - Right Hand Upper Targeting Vane Consecutive Feedback Fail, Channel A
NA59 - FHFTVUX - Right Hand Upper Targeting Vane Consecutive Feedback Fail, Channel B
```

Fig. 4 Nozzle schematic display.



Flight Conditions

Altitude	54730	Airspeed	191.9
Mach No.	0.25	IFPC Mode	CONV
Alpha	52.37		

IRIG 000:00:00:06
Quality 16
FISAs lost 2066
FISAs lost 2161

Increase Airspeed
First Attempt
Flight

PLA	1.78
TLA	61.55
N1	53.9
N2	48.2

PLA	48.69
TLA	61.55
N1	53.9
N2	40.2

Fig. 5 Propulsion expert system display.

The expert system inferencing is run as a background process that communicates with the foreground display controller. This is similar to the judge model described by McCauley.[4] The display controller performs the mundane task of keeping the display up to date while the judge sits in the background making executive decisions. When a significant event occurs, the judge informs the display controller who then updates the display messages in the center of the screen. The lower message describes the status of the engine and the upper message indicates procedural information that the pilot should implement.

The rules recognize several engine states and potential engine states. The normal status of the engine would be displayed in green as "taxi" or "flight." However, when an air start is attempted, the rules will indicate that the aircraft is in a proper envelope for doing so. During the start attempt, the rules may indicate (for example) that a potential hot start is in progress by displaying an appropriate message in yellow. If an actual hot start occurs, this will be displayed in red. If the start occurs properly, this will be indicated by a green "good start." One engine state, stall, is easy to detect, but methods to predict it are elusive and may be something outside current technology and expertise.

The knowledge base was originally formulated by the knowledge engineer using a knowledge matrix similar to that described by Braun.[5] Presenting this to the expert encouraged the expert to develop a flowchart of what monitoring activities were desired. The rules were then written combining these different representations of the knowledge base. At that point, the continuous iteration process of test, evaluate, and modify began.

One aspect of the rule structure that needs constant attention is that of sequential dependency. There are portions of the rules that only execute successfully when a known sequence of events occurs. For example, during an air start if an unsuccessful start occurs there are a series of methods used to reattempt a start. These methods, according to the book, should be tried in a certain sequence. If, in actuality, they are not, then the rules become lost. Ideally, the rules should be able to recognize any given situation without recourse to remembered conditions or sequence. The existence of this sequential dependency is partially due to the actual sequential nature of engine events, but is also due to the inexperience of the expert with the concepts of rule-based expert systems. As discussed later, experts have to be educated to expert system concepts. As shown in the following two examples, the iteration process rapidly indicated where some of the sequential dependency could be eliminated. (The rules given in the examples do not conform to CLIPS syntax and only show conditions and actions relevant to the example.)

Example 1: When the expert system is initiated, the engine status is set to unknown. The original assumption was that the monitoring would always start with the engine off. Datawise, engine off is indicated by the power lever angle (PLA). As such, rule 1 was written,

```
if (PLA << 15)
and (engine status = unknown)
then (engine status = off)
```

The concept was that once engine off was established, monitoring could begin. Unfortunately, the assumption that monitoring always starts with the engine off is false. Since almost all other rules required that rule 1 be fired first, the falsity of this assumption blocked the rules from ever firing. The status of the engine was left forever unknown.

The solution to this was rather simple. Add the rule:

```
if (engine status = unknown)
and (weight on wheels)
then (engine status = taxi)
```

Then change rule 1 to rule 1a:

```
if (PLA << 15)
and (engine status = unknown or taxi)
then (engine status = off)
```

This not only eliminated a sequential dependency, but brought awareness of the omission (from the rules) that the engine can go from taxi to off.

Example 2: The following rules define the initial sequence for a start attempt (*N*2 is core speed percentage).

Rule 2:

```
if (engine status = off)
and (N2 << 5)
then (engine status = jfs start attempt)
```

Rule 3:

```
if (engine status = jfs start attempt)
and (5 << N2 << 18)
then (engine status = awaiting N2 criteria)

if (engine status = awaiting N2 criteria)
and (N2 >> 18)
then (engine status = start attempt)
```

Although this is, in fact, the sequence the engine goes through, the value of *N*2 may jump from 4 to 20 between successive rule firings. When this happens, the rules are hung at rule 2. By eliminating the condition in rule 3 of (*N*2 << 18), the rules will fire properly. This does not eliminate the sequential firing of the rules but it does remove the dependency on the sequentiality.

The first example illustrates the trap of hidden assumptions. In this case, even when actually stated the assumption, in and of itself, it does not seem to be in error. Only when considered in a broader scope or when testing does it become clear that the assumption is a major stumbling block for the rules. The second example also has a hidden assumption: the data is continuous. Although the physical phenomenon, core speed, changes in a continuous fashion, the data seen by the expert system are discrete samplings.

Both of these examples illustrate the difference between procedurally developed monitoring structures and the true activities of an expert. An expert can certainly come in during the middle of a test and understand the situation. Further, an expert does not watch all parameters all of the time—an expert takes discrete samplings when necessary.

Another difficulty with the rules is that of noisy data. When noisy data is input to the expert system, it gets lost fast. Two partially successful methods for dealing with this have been implemented. The first is to provide the mouse activated reset button in the top right of the screen, which starts the rules from scratch. This method works only in the absence of sequential dependency. The second solution is to filter the data. Since noisy data is a long standing issue in telemetry monitoring, digital filtering has become a large field of study. The scope of the development effort has not included this analysis so that only a primitive filter has been implemented.

This expert system was written specifically for the STOL, but the concepts apply to most aircraft. What would it take to modify this for another aircraft? The engine diagram would have to be modified slightly since few aircraft have vectoring nozzles. Most of the gauges are generic, but a different engine might have some parameter of particular interest, and so removing or adding a gauge might be necessary. That is, some minor changes to the display would be normal. What about the rules? The process an engine goes through when starting is fairly well understood, and so the basic structure of the rules probably would not change much. One area of change would be limits. (What PLA indicates engine off?) More important, though, the procedures followed in the case of a failed start are usually different for different aircraft. Thus, the procedural information and recovery processes would have to change. That is, although the existing rules could serve as a solid model, a fairly careful rework of the rules would have to take place. Hopefully, though, the more aircraft supported by this system, the less work necessary for adding another.

Knowledge Engineering

The process of writing computer programs is referred to as software engineering. The process of developing a knowledge base is referred to as knowledge engineering. The creation of an expert system requires an intimate cooperation between these two disciplines. However, there is a conflict in the respective processes. Software engineering tends to be a very sequential process. Requirements analysis, design, coding, and testing are done in that order. Many studies have shown that not to do these in sequence produces poor and costly software. Knowledge engineering engages in these activities as well, but the iterative process is very rapid—rapid enough that it becomes an underlying theme in support of other activities and is preferably noticed only as a consequence of marriage with software engineering. The conflict arises from the need to do several activities continuously and simultaneously in contrast to singly and sequentially. To illustrate this point, several knowledge engineering activities are discussed.

Learning the Expert's Language

Knowledge engineers tend not to be domain experts (the domain being the area of concern for the expert system). During the development of the applications noted earlier, the knowledge engineers had to become familiar with the real-time flight testing environment and the STOL aircraft. One approach to learning the technical terminology used by the experts was reading software specifications and design documents. This was a good starting point, but it is possible to get bogged down in too much documentation. A much better approach was conducting interviews with the domain experts. Sitting down and talking with someone is the best way to learn their language.

When programming, it is helpful for a software engineer to know something about the subject matter being programmed. However, it is possible to isolate code modules, such as equations or data manipulations, and program them without knowing their relation to the subject matter. In contrast, when trying to emulate the object manipulation of intelligent inferencing, it is not feasible to completely isolate these objects. It is necessary to comprehend the relation of these objects to the other objects in the system. An understanding of where these objects fit into the big picture is necessary.

Teaching the Expert About Expert Systems

To domain experts, the term expert system is usually an unfamiliar buzz word. As such, knowledge engineers have to educate as well as be educated. This process necessarily requires many discussions and presentations. However, a very successful technique is the use of prototypes. Domain experts are most stimulated to learn when presented with working prototypes (no matter how primitive). Prototypes provide the experts with tangible illustrations of the capabilities and applications of expert systems. Prototype demonstrations not only create an interest in the applications of expert systems, but allow experts to raise their understanding and expectations of expert systems to a level commensurate with achievable goals.

This aspect of knowledge engineering is not entirely in conflict with software engineering. Software engineering often requires working with nonsoftware people so that education to software techniques occurs during project development. However, it is possible to produce usable software for someone who has never touched a keyboard. In contrast, the domain expert must actively pursue the limits of expert system technology to produce a useful tool. Keyes[6] provides an example from the financial industry.

Producing the Man-Machine Interface

The main thrust of knowledge engineering is building a knowledge base, but a knowledge base is useless without a means of communicating the knowledge to the users. For real-time communication, intelligent graphics are essential.

Traditionally, pictures are a vital part of engineering documentation—and for good reasons. Pictures provide a symbolic representation of relations between different aspects of a given system. They also provide a tangible model people can point to when discussing abstract or obscure aspects of a complex structure. Similarly, intelligent color graphics provide a direct visual link to the mind's cognitive model. In this sense, graphical layouts form a critical part of the knowledge base. A technical diagram incorporates a great deal of information and, very likely, contains more detail than a mental picture. By adding dynamics to the graphics, a rapid communication method is created. All of this is especially true for a real-time expert system. Reducing man-machine communication time from the minutes of paper lookup to the virtually instantaneous icon color changes is imperative when real-time means fractions of seconds. Although rules and pictures often perform different functions, there is enough of a functional intersection that it is reasonable to say: A picture is worth a thousand rules.

Since the knowledge base is constantly changing as it develops, and since the graphical interfaces form an important part of the knowledge base, a distinct conflict arises with software engineering. To follow the software engineering process would require a relatively static design for the interface. Since this is counterproductive to knowledge engineering, a compromise needs to be made.

Identifying Particular Objectives

Whenever an expert system project is started, general objectives are put forth. However, it is up to the knowledge engineer to produce specific detailed objectives. This cannot be done up front. As discussed earlier, the knowledge engineering process is as much one of learning as it is of producing a knowledge base. The useful end product may not bear much resemblance to initial designs. Perhaps this will not always be so, but the field is young enough that there are not many experts on expert systems. This forces the process to include true research—proposing directions and solutions, then testing them to see if they apply to the actual situation—which requires a flexible and dynamic environment.

This is in direct and complete opposition to the software engineering process. To produce cost-effective software, requirements and designs must be virtually frozen before coding begins. This conflict is not impossible to overcome, but awareness is needed to do so.

Melding Two Disciplines

The essential conflict is between sequential and continuous activities. The activities discussed earlier are necessarily ongoing—they never stop and are all happening at the same time. The chances of learning another discipline in 6–12 months are minimal, and the extreme learning curve prohibits static detailed design. The general solution is to provide flexible software tools and an architecture that isolates the changing products of knowledge engineering.

The three-layer design described earlier is one approach to minimizing this conflict. Isolating the data acquisitioning allows a large part of the support software to be developed with no impact on, or from, the knowledge engineering process. Similarly, separating the truly heuristic knowledge base (layer 3) from the informational and graphical knowledge base (layer 2) provides the flexibility of modular development. Additionally, concentrating software requirements on graphics tools that allow quick modification of displays lets the software effort aid the knowledge engineering without being hindered by it.

Conclusions

Developing a real-time expert system requires a large and dependable support base. Commercial off-the-shelf hardware and modern data acquisitioning software are currently available to provide this support.

Graphical displays are not often enough thought of as part of the knowledge base. However, an initial requirement for an expert to act real time is to be a fast data retriever. The expert must be able to interpret computer displayed data rapidly in order to make quick decisions. Intelligent graphics incorporate large amounts of knowledge and provide a significant decrease in the expert's activity as a data retriever.

Providing a solid support base as well as an efficient, flexible man-machine communication method allows for successful knowledge engineering. Knowledge engineering is, however, a fledgling discipline that is still overcoming inherent conflicts with its necessary partner—software engineering. By recognizing the large learning curve and encouraging intimate interaction between the expert and developers, these conflicts can be removed and useful real-time expert systems produced.

Acknowledgments

Thanks must go to Arthur N. Rasmussen, Mitre Corporation; Robert V. McFarland, UNISYS Corporation; and to the following people from Edwards Air Force Base: Catharine Hoaglund, Carlos Diaz, Gerald Brandt, Dave Robertson, Phillip B. Arnold, Larry Smith, Archie Amos, Henry Bunch, and William Kitto. Very special thanks go to John F. Muratore, NASA Johnson Space Center. Also, a heartfelt appreciation goes to Susanne Weiss, DLR, Braunschweig, West Germany, for her work on the propulsion expert system.

References

[1]Heindel, T. A., Kindred, E. D., Madison, R. M., McFarland, R. V., Muratore, J. F., Murphy, T. B., Rasmussen, A. N., and Whitaker, C. L., "Real-Time Expert System Prototype for Shuttle Mission Control," Systems Operation Automation and Robotics Conference, July 1988.

[2]Muratore, J. F., "Trends in Space Shuttle Telemetry Applications," *Proceedings of the International Telemetering Conference*, Instrument Society of America, Research Triangle Park, NC, Vol. 23, International Foundation for Telemetering, 1987, pp. 11–16.

[3]Disbrow, J. D., Duke, L. D., and Ray, R. J., "Preliminary Development of an Intelligent Computer Assistant for Engine Monitoring," NASA Rept. TM 101702, Aug. 1989.

[4]McCauley, B., "Expert Systems in Data Acquisition," *Proceedings of the International Telemetering Conference*, Instrument Society of America, Research Triangle Park, NC, Vol. 23, International Foundation for Telemetering, 1987, pp 41–49.

[5]Braun, R., "Expert System Tools for Knowledge Analysis," *AI Expert*, Vol. 4, No. 10, 1989, pp. 22–29.

[6]Keyes, J., "Where's The 'Expert' in Expert Systems?," *AI Expert*, Vol. 5, No. 3, 1990, pp. 61–64.