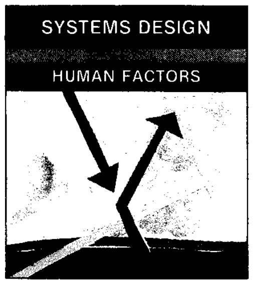
Computer-aided fighter pilots

How an interdisciplinary team overcame technical and organizational hurdles to develop an expert system that tailors its real-time responses to a pilot's flying style



In late 1984, a disagreement emerged among subcontractors and members of the Lockheed Corp. team that was preparing to bid on a U.S. Air Force and Defense Advanced Research Projects Agency (Darpa) contract for an artificial-intelligence (AI) project to aid fighter pilots in flight. Assembled specialists from the company's Marietta, Ga., facility—from electrical engineers to psychologists to fighter pilots—were split on which design approach to take. Should the system, called the Pilot's Associate (PA), be a conventional expert system, recommending a course of action from a fixed knowledge base about tactical air combat? Or should it take the novel step of tailoring its expert response to the pilot's particular style of flying the aircraft?

Rather than deciding by fiat, a Lockheed manager divided the group into opposing sides. We at Search Technology Inc., Norcross, Ga., led what was called the blue-sky team, which advocated a system specifically tailored to a pilot's needs. Our competition, calling for a more conventional approach, was the down-to-earth team, which proposed adding expert-system techniques to an existing avionics architecture for communicating among on-board computers as well as sensors and actuators.

In our blue-sky team presentation to Lockheed, the first slide that represented the system we planned to develop showed a knight on horseback alongside the R2D2 robot of *Star Wars* fame, symbolizing a pilot accompanied by an automated helper. Subsequent frames showed how the system would use what we called an intent interpretation software module to gather data about stick movements, radar, and equipment sensors to infer in real time the intent of the pilot's actions. This information could then be employed to compare intent with what was actually happening in and around the plane.

Our approach, interpreting pilot activity as a basis for applying expert-system recommendations, was a significant factor in Darpa's and the Air Force's awarding of the AI project to the Lockheed team. In addition to our group, another team of McDonnell Douglas Corp., St. Louis, Mo., and Texas Instruments Inc., Dallas, was awarded a contract to develop software prototypes for a competing version of the PA.

New challenges

Moving from a series of presentation transparencies to a system that worked in real time, however, encompassed far worse design challenges. One of the primary obstacles—and the reason for the intent interpretation module we proposed—was the need to adapt expert-system technology to perhaps the most critical of any real-time environment, a fighter cockpit. A system expected to give the pilot immediate information about threats and targets was markedly different from, say, a computer-based system that "interviewed" a physician question by question to elicit a disease diagnosis. Not only does it need to communicate rapidly with the pilot, but it must suppress unnecessary details so that the pilot can focus on what is truly important.

The scope of the project also made it stand out. Many expert systems succeed by narrowing their knowledge base to a carefully circumscribed area of expertise, gained through lengthy interviewing of individuals prominent in a field. For instance, an electric utility might develop an expert system to help replace a retiring staffer who has mastered the intricacies of repairing hydroelectric dams.

But the PA, besides the pilot's inputs, required in its knowledge bases expertise from human-factors specialists to decide how the information should be displayed, psychologists to establish benchmarks for pilot performance in combat, and engineers to define the relevant operational characteristics that need to be monitored by the expert system.

The project's breadth can be seen in its multiple software modules, all of which had to work together: a tactics planner, crafted by ISX Corp. (formerly Teknowledge Federal Systems), Thousand Oaks, Calif., that supplied recommendations on tactical maneuvers such as attacking or evading; a mission planner created by Lockheed that recommended a flight path; General Electric Co.'s system status module that monitored engines for disabled components, overheating or vibration, and engine fires; a situation assessor, the work of various project contractors, that evaluated targets and threats; and a mission manager from Lockheed containing a blackboard, or memory space, that coordinated communication among the various software modules.

Search Technology, which had been working with Lockheed a year prior to the contract award, was retained by Lockheed as the subcontractor for yet another module, the pilot-vehicle interface (pvi) that was to perform intent interpretation and determine how much information to display to the pilot at a given moment [Fig. 1].

William B. Rouse, Norman D. Geddes, and John M. Hammer Search Technology Inc.

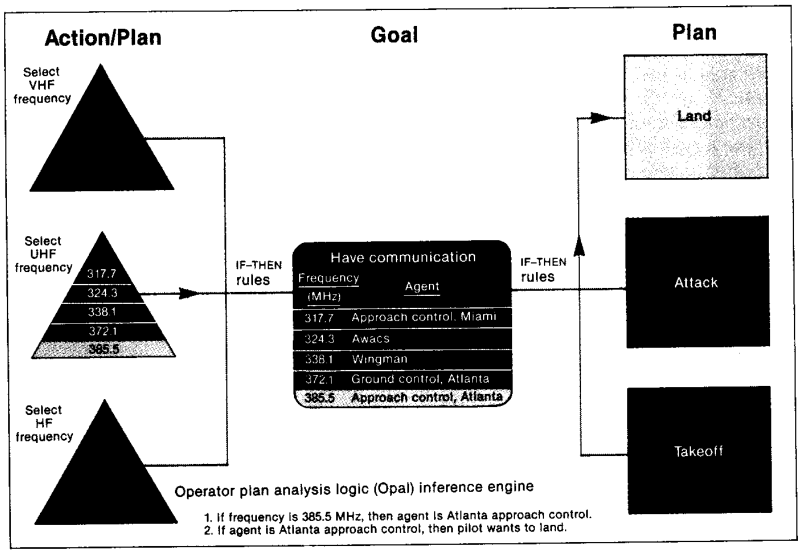
Defining terms

IF-THEN rule: in the context of an expert system, a statement declaring that a particular condition is to be followed by a particular action; it is also called a production rule.
Inference engine: the part of the expert system that selects and executes rules from the knowledge base.
Knowledge base: the part of an expert system that contains the IF-THEN rules that represent a human expert's understanding of a domain, the problem area covered by an expert system; it also includes knowledge about events, objects, and situations.
Plan and goal graph: a hierarchical graph of plans and goals. The lowest level consists of nodes in which an action—a type of plan—a pilot takes is used to arrive at goals by executing an IF-THEN rule that moves up the graph's plan and goal nodes as it infers what the pilot is trying to do.

In operation, if a pilot deviates from the tactical attack plan—turning the radar off and moving away from a target or threat, for instance—the pvi brings up displays with relevant defensive systems, like chaff or electronic countermeasures. Conversely, if the pilot focuses all attention on a combat maneuver, the pvi is to sense this and refrain from bothering him with malfunction warnings that could be delayed until later. Once the pvi knows what the pilot wants to do, it can detect flying errors, determine how serious they are, report them to the pilot, and possibly correct them if the pilot has authorized it to do so. The pvi is also the interface between the pilot and other PA system modules, such as the tactics and mission planner.

The need to manage so much information simultaneously stems from the growing density and efficacy of both airborne and land-based threats and the increasing complexity of fighter countermeasures to respond to those threats. In the F-18, there are over 40 radar modes, more than most pilots know how to use.

Technically, the pvi employs a combination of AI and algorithmic processes. It consumes more code than any other PA system module, combining 800 rules with an intricate tree-like structure that is used to infer a larger intent from a simple action: does moving the stick, for example, mean that the pilot wants to engage or evade a threat?


Action/Plan
Goal
Plan
Select VHF frequency
Select UHF frequency
317.7
324.3
338.1
372.1
385.5
Select HF frequency
IF–THEN rules
Have communication
Frequency (MHz)
Agent
317.7 Approach control, Miami
324.3 Awacs
338.1 Wingman
372.1 Ground control, Atlanta
385.5 Approach control, Atlanta
IF–THEN rules
Land
Attack
Takeoff
Operator plan analysis logic (Opal) inference engine
1. If frequency is 385.5 MHz, then agent is Atlanta approach control.
2. If agent is Atlanta approach control, then pilot wants to land.

[1] The pilot-vehicle interface (pvi) consists of expert-system software modules that interpret a pilot's actions, such as flipping a switch, and relays them via an input decoder to a plan-goal graph (not shown) in the intent interpretation module. The plan-goal graph is used to infer the pilot's intent, information which is passed to the interface manager or to three other modules: the error monitor, which checks for flying mistakes; the resource model, which estimates the pilot's mental workload; and the performance model, which predicts how long a pilot will take to complete a task. Adaptive aiding determines whether the pilot needs automated assistance from inputs from the error monitor, the resource model, or the performance model. The interface manager decides what display information the pilot needs before going to the display generator outside the pvi; a blackboard lets modules share information.

A fast pace

As with any other expert system, the early part of our work was directed toward establishing a knowledge base: data from experts that could then be processed by the system's inference engines. The best expert sources were those with cross-disciplinary specialties. For instance, one team member, a cognitive psychologist who helped work out specifications for a knowledge base of in-flight performance, was hired in part because he had experience as a nuclear engineer.

Psychologists generally prefer to evaluate hypotheses by conducting experiments, an approach that would have been out of step with the fast pace of this program. Drawing on his experience as an engineer, the psychologist on our team understood the need to proceed with a design despite gaps in performance data.

Pilots, of course, were a key part of this most labor-intensive portion of the project. To find out whether fighter pilots would really use the functions we were planning to design into the pvi, we interviewed some.

One question we explored was whether the pilots would be willing to have the PA take over for them. Nine of the 10 interviewed said that the notion of the plane routinely taking control was anathema to them. But when asked whether they would mind the plane's taking over if they were unconscious and in a tailspin, all except one were amenable. That holdout said that even the possibility of being rescued from certain death was not enough to yield the control that he believed should remain the pilot's prerogative. Reacting to a more middling scenario, most said that if an engine stalled while engaged in combat, having the pvi restart the engine would be welcome.

The pilot's intent

With these raw inputs, we then had to tackle the software design so the system could make expert recommendations without stopping to quiz pilots during air-to-air combat. Having a knowledge-based system interpret a pilot's actions was achieved by defining the natural language the pilot and aircraft used to communicate with each other. The sentences and paragraphs of that language were the numerous switch activations, stick movements, and engine and radar status readings that made up the operation of the aircraft; these were presented to the pvi through a software interface module called the input decoder.

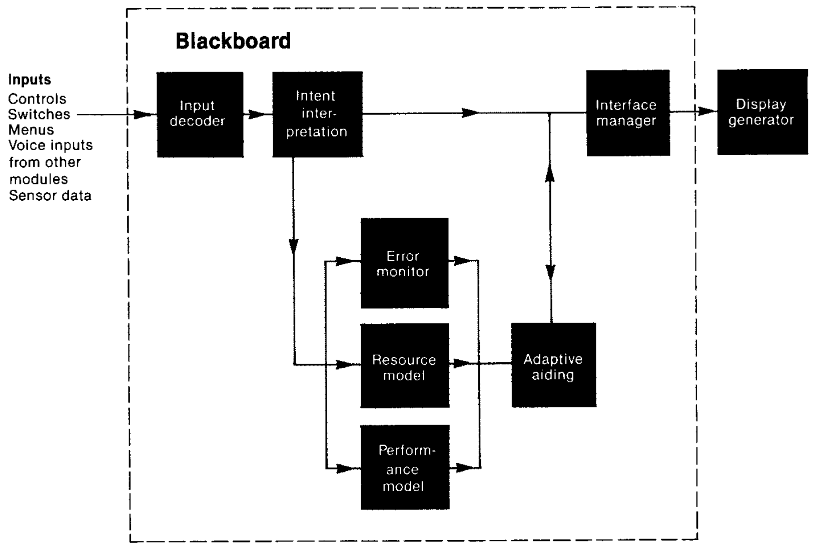
It was our job—and perhaps one of the biggest design hurdles—to take those raw inputs and create a software structure that could infer from them an understanding of the pilot's intent. The key to the system architecture was the plan-goal graph (pgg), which describes the elements used to link a pilot's actions, or plans, with a particular mission goal [Fig. 2].

Influencing the design of the pgg was the work of Roger Schank, a professor at Northwestern University, Evanston, Ill. Schank programmed software with a plan-to-goal structure that drew conclusions about activities in a restaurant. How satisfied a restaurant-goer was with a meal or service was inferred from IF-THEN rules that were applied to the details of a written description of the meal.

Search Technology's pgg consists of 300 nodes, each representing a plan or goal. An action—a change in the state of an aircraft system that may result from, say, a pilot's flipping of a switch—is a type of plan that is represented as the lowest node of the pgg. The goal of the action—one node up the pgg—is then inferred from the plan. Plans and goals alternate in a chain up the graph. Links among graph nodes are established using some of the 800 IF-THEN rules from the pvi's knowledge base. The software that activates these rules is an inference engine called Operator Plan Analysis Logic (Opal), which selects IF-THEN rules associated with a particular pgg node until it finds one that works. In this way, it searches up the graph from plan to goal.

One simplified example illustrates how the pgg figured out the pilot's intent to land the plane: IF the pilot takes the action of

[2] Determining a pilot's intent is done by a plan-goal graph, which links plan and goal using rules from the pilot-vehicle interface's inference engine, Operator Plan Analysis Logic (Opal). An action, a type of plan, is represented by the pilot turning on the UHF radio to a particular frequency in the triangles at left. Opal then executes IF-THEN rules until it finds one that infers a goal: "have communication" with the Atlanta approach controller, indicated by the frequency 385.5 megahertz. The goal, in turn, leads by way of another IF-THEN rule to a higher node, a plan that infers from the agent (here, the Atlanta approach controller) that the pilot wants to land in the Atlanta area. Some frequencies were chosen only for illustration and are not the real ones used.



turning on the fighter's UHF radio at a frequency of 385.5 megahertz, THEN Opal infers the "have communication" goal. That frequency corresponds to the goal of establishing contact with the Atlanta approach controller. Then, proceeding up the graph with another rule: IF the goal is to contact Atlanta and IF the pilot is near the city, THEN the pilot wants to land.

The plans and goals may consist of a script—for example, a set of formal procedures to be taken for an engine fire emergency: pull fire handle, dump fuel. Or they may be loosely ordered sets of alternatives, particularly in combat, because of the impossibility of proceduralizing everything that may occur in flying a jet fighter.

The pvi incorporates a feedback mechanism designed to cope with the volatile environment of the fighter cockpit. This AI concept is called successive disambiguation of uncertainty: if the pgg has difficulty determining a pilot's intentions, then Opal will reach a tentative conclusion by making the closest match between plans and goals. But, if the match is not entirely consistent, the system will correct its interpretation after observing subsequent plans that could be interpreted as having a differing goal. A pilot may turn off the radar and the pvi may conclude that the goal is to launch a "stealthy" attack on a target. However, if a moment later, the pilot moves the stick to turn away from the target, evasion would be the revised goal.

A reconciliation

At times it was necessary to reconcile differing design needs. After noticing that we at Search Technology and the tactics planner design team were each developing our own plan definition, we decided to work toward a common pgg. One major difficulty was reconciling the fact that ISX needed to keep many pgg nodes in one place—those for radio communications, for example—while we wanted to spread them out. For tactics planning, having nodes for different radio frequencies together made it easier for the software to control the radios. By contrast, we needed the frequencies spread out throughout the pgg because the reason for using a radio might be anything from a combat maneuver to reporting an engine malfunction. The final pgg blended the two approaches.

Once the intent-interpretation software had done its work, another module—the error monitor—searched for anomalies in the pilot's actions. If some action could not be explained, the error monitor searched for a reason. For instance, if the pilot suddenly tuned the radio to the Atlanta approach frequency when the plane was, in fact, flying near Miami, the error monitor would search its knowledge base for similar frequencies, an effort that might succeed in locating and flagging a mis-entered digit.

The error monitor was built on five years' earlier work that we did for the National Aeronautics and Space Administration's (NASA's) Ames Research Center, Moffett Field, Calif., in which we studied pilot performance for civilian aircraft. From this, we were able to identify some of the subtleties involved in causing a pilot to make a mistake in flying an aircraft. One thing we learned was that a pilot faced with an aircraft malfunction such as an engine failure might unconsciously revert to a procedure for landing that presumed all engines were operating. This occurred particularly when the two procedures were very similar. Cognitive psychologists describe a pilot who makes this error as being "captured" by the routine way of doing things. The error monitor was programmed to look for and flag such deviations.

When an error is located during flight, it is passed to yet another module, adaptive aiding, with a recommendation for corrective action. Adaptive aiding must then decide what information to pass along to the pilot. It might not inform him of a possible error, however, if he is engaged in a critical but unrelated activity such as air-to-air combat. In this case, adaptive aiding would automatically correct the error, but only if the pilot had authorized such an action before the flight.

Two other pvi modules also communicate with adaptive aiding: the resource model, which tries to predict whether a pilot could devote attention—vision, hearing, and hand movement—to a particular task, and a performance model, which predicts how long it would take to complete a task.

Formatting information for display—where hydraulic-system status, an error message, or other information appears on the screen—is the job of the interface manager. The information itself might come from the aircraft or any of the PA modules.

Both Darpa and the Air Force specifically directed us not to turn the PA into a display-design program. Although the interface manager only passes information to the screen-driver software developed by Lockheed, called the display generator, Search Technology was asked to take part in selecting which display technologies to use. We were told to concentrate on utilizing existing systems, such as touch screens, color cathode-ray tubes, and voice input/output systems.

Ways to communicate

A major difficulty in developing the PA was finding effective ways to communicate—both among human designers and among the software modules themselves. Communication was problematical because it occurred at a more abstract level than the avionics of a conventional aircraft. For example, instrumentation on most aircraft senses airspeed. The dependence of speed upon altitude, which defines the meaning of airspeed, is something described in elementary aerodynamics textbooks.

But there was no textbook to which we could turn to obtain a quantitative definition of how dangerous a threat was. While an enemy might be tagged with a threat value of from 1 to 100, the threat's myriad dependent variables affect the extent of the threat: whether the enemy is aware of the pilot's aircraft is just one dependent variable that governs threat severity.

Because of the complexity, we employed a variety of programming and design concepts, some borrowed from areas outside the AI field, including operations research. Although the software was programmed in Lisp, we elected not to use standard expert-system shells, such as Knowledge Engineering Environment or Acquisition of Strategic Knowledge, because their software overhead would be unacceptable in a real-time avionics environment.

The actual designing and programming of the pvi on a Symbolic 3640 workstation and three IBM PC/ATs proved an exercise in tedium. One team member commented that the first 5–10 percent of knowledge engineering is enjoyable because demonstration software can be brought up quickly showing how the system might work in a limited scenario. The remaining work proves arduous because of the many interactions that can occur in such a tightly structured system.

Once we had collected, compiled, and represented the information in the knowledge base for our first prototype, we had to start all over because our understanding of the problems involved had broadened. Redesigning the pgg or adding a new plan also required adding supporting knowledge to the other modules. Including a target-recognition device that identified an enemy fighter from its exhaust emissions required painstaking manual changes to many pgg nodes.

We still desperately need tools to assist in the creating of knowledge bases. This might be a software utility that would simulate the aircraft's altitude, sensors, and weapons, and then execute new design rules to show what impact they have on a particular plan in the pgg. It would flag a knowledge-base error if the program incorrectly interpreted a pilot's action and would allow the system's knowledge to be edited.

Search Technology did develop a simple tool that allows pgg and display debugging, testing, and editing. Without that tool, we would have had to modify elements of the knowledge base with a text editor and then use Lisp program stepping and tracing facilities to check program execution. The Lisp facilities are too slow to be practical in debugging large-scale knowledge bases.

Rapid prototyping—a technique rarely used on military projects—allowed us to squeeze a complete cycle of requirements specification, design, and coding into a four-month period. One technique was to look ahead to the most important design goals for each module for the next prototyping cycle.

We also speeded up the software documentation process. If we had been required to produce fully documented, deliverable software for each prototype, we would probably still be on the second or third round, rather than the ninth prototype. Fortunately, the Government's streamlined approach dropped the requirement of extensively documenting the software before it was written. Instead, details about what happened during the development process were delivered after the completion of each cycle.

Foraging for data

Hanging over the entire project was the fact that through all the prototypes we knew relatively little about the mid-1990s aircraft for which we were designing. Lockheed, one of the two prime contractors for the Advanced Tactical Fighter (ATF), the model for the aircraft of the new decade, was able to release only scant details of the classified, competition-sensitive system it was designing. Even so, much time was spent in scrounging for information about technologies likely to be used on the ATF—electronically steered radar, for one. It would have been far simpler to take technical data from the F-15 or another fielded aircraft in order to design the software in such a way that it could be adapted to the expected technological advances.

Because of this gap in our knowledge, we had to make assumptions about the aircraft. Future sensors are likely to be better, but by how much? If the fighter's sensors are assumed to be too good, the simulator could account for location, velocities, and composition of enemy forces, giving the pilot an unrealistically comprehensive overview of his tactical situation.

The new electronically steerable radar antennas may be directionally agile, eliminating the need for continuous scanning or steering. The pgg has to reflect the new search modes, even though current pilots have no operational experience with them. The only certainty we had to go on was that in every situation, each fighter had to be able to cope with the chance of being outnumbered.

The early phases of our work emphasized development over testing. But we have not had enough time to learn how the system behaves as an ensemble. So we have yet to fully explore the limits and accuracy of Opal's ability to predict a pilot's intentions. To test these types of scenarios and study how a complex expert system might produce erroneous conclusions, Search Technology is under contract with NASA's Langley Research Center in Hampton, Va. We are trying to determine whether the system could produce a false conclusion because of data it lacks in its knowledge base. One consequence: the pvi might attempt to substitute an inappropriate screen display.

Although the PA started as a high-risk technology project, it is nearing the point of becoming an integrated prototype for missions involving offensive, counter-air attacks, targets of opportunity, and strike escort. Because we were able to beat Darpa's deadlines and technical expectations, the program was redirected in 1988 toward near-term development. This second phase of the program, which began in the fall of 1989, has the goal of making the entire system run in a real-time procedural language (Ada or C) on avionics processors.

The prototyping and design concepts developed by the PA team may have broader application beyond aiding fighter pilots, possibly in such areas as flying commercial aircraft and air traffic control procedures. Apparently, Darpa and the Air Force believe the program has done well enough for them to consider early infusion of this technology in several forthcoming demonstration aircraft programs.

To probe further

Papers presented at the annual IEEE Systems, Man and Cybernetics Conference often deal with human-machine interaction. The 1990 conference is scheduled for Nov. 4–7 at the Sheraton Universal Hotel in Los Angeles. For information, call conference coordinator Amos Freedy at 818-884-7470.

A more in-depth description of the human-factors engineering behind the pvi can be found in "An Architecture for Intelligent Interfaces: Outline of an Approach to Supporting Operators of Complex Systems" by William B. Rouse, Norman D. Geddes, and Renwick E. Curry, *Human-Computer Interaction*, 1987-88, Vol. 3, pp. 87-122; Lawrence Erlbaum Publishers, Hillsdale, N.J.

About the authors

William B. Rouse (F) is chairman and chief scientist for Search Technology Inc., Norcross, Ga., which he cofounded in 1980. He has been a professor of industrial and systems engineering at the Georgia Institute of Technology in Atlanta, and of mechanical and industrial engineering at the University of Illinois in Urbana.

Norman D. Geddes, a former Navy attack pilot and instructor, is a Search Technology consultant. His doctoral thesis at Georgia Tech, completed in July, dealt with intent inference.

John M. Hammer (M) is a principal scientist at Search Technology and program manager for the pvi and director of the Operator Support Systems Group, which researches and develops applications for intelligent interfaces. ◆