

Pilot's Associate

A Cooperative, Knowledge-Based System Application

Sheila B. Banks and Carl S. Lizza, Wright-Patterson Air Force Base

TODAY'S FIGHTER PILOTS FLY complex aircraft to perform complex tasks. These pilots must be keenly aware of external situations — threats, for example — as well as the internal status of aircraft systems, and options for dealing with faulty equipment. Intricate aircraft systems increase the need for intelligent cooperation between pilots and aircraft. An additional requirement for intelligent systems designed to aid pilots is real-time operation in highly dynamic situations.

The Pilot's Associate program is a joint effort of the Defense Advanced Research Projects Agency and the US Air Force, managed by the Air Force's Wright Laboratory. The program began in February 1986 as an application demonstration for DARPA's Strategic Computing Initiative. DARPA wanted to advance the program's technology base, principally in the area of real-time, cooperating knowledge-based systems. The Air Force wanted to explore the potential of intelligent systems applications to improve the effectiveness and survivability of post-1995 fighter aircraft.

The Pilot's Associate concept developed as a set of cooperating, knowledge-based subsystems: two assessor and two planning subsystems, and a pilot interface. The two assessors, Situation Assessment and

System Status, determine the state of the outside world and the aircraft systems, respectively. The two planners, Tactics Planner and Mission Planner, react to the dynamic environment by responding to immediate threats and their effects on the prebriefed mission plan. The Pilot-Vehicle Interface subsystem provides the critical connection between the pilot and the rest of the system. The interface ensures that the system as a whole provides the information the pilot wants, when it is needed.

The Phase 1 Pilot's Associate program contained two distinct efforts. One focused on air-to-ground mission scenarios and platforms, and was developed by a team directed by McDonnell Aircraft Company. This article focuses on the air-to-air subsystems developed by a team led by Lockheed Aeronautical Systems Company. We

participated in both development teams as the Air Force's program and technical management.

The first significant program demonstration, Demo 2, was scheduled for two years into the program. This was intended as a nonreal-time, technical demonstration of a complete but skeletal Pilot's Associate system. The objective, in other words, was to demonstrate an architecture for cooperating knowledge-based subsystems in a system with limited depth of knowledge and capability. An operationally impressive system was not anticipated during the early stages of development. However, Demo 2 gave us a glimpse of the potential usefulness of Pilot's Associate in the complex air-combat environment.

Phase 1 efforts culminated in Demo 3, originally scheduled for three years into

ARTIFICIAL-INTELLIGENCE TECHNOLOGY CAN PLAY AN IMPORTANT ROLE IN THE DYNAMIC, COMPLEX DOMAIN OF FIGHTER AIRCRAFT. THIS ARTICLE EXPLORES THE LESSONS WE LEARNED IN BUILDING A COOPERATIVE, KNOWLEDGE-BASED SYSTEM TO HELP PILOTS MAKE DECISIONS.

the program. As a result of Demo 2's success, the program was redirected to apply Pilot's Associate technology to a current or planned service aircraft and to emphasize near-term computing hardware. Rescheduled to the fourth year due to this redirection, Demo 3 showcased a functional prototype intended to execute in nonreal time. In addition to the architecture for cooperating knowledge-based subsystems required at Demo 2, the Demo 3 system contained the functional prototype capability envisioned for Pilot's Associate. However, the Demo 3 prototype still lacked the knowledge depth required for full functioning.

Phase 2 is now under way, and a fully functional system executing in a real-time, man-in-the-loop environment should be completed in 1992. Looking forward to this goal, Phase 1 also included system and subsystem analysis to determine both hardware and software requirements for a fully functional real-time system.

The first two years: Demo 2

The Lockheed/Air Force team successfully completed the first milestone, Demo 2, in March 1988 in a laboratory cockpit simulation driven by six Symbolics machines and a VAX. Because Demo 2 was a nonreal-time demonstration, the team also generated a real-time mission replay to allow visualization of the final Pilot's Associate product. We conducted the live, nonreal-time laboratory run using the same mission, which allowed system interaction.

The scenario. The Demo 2 mission scenario was a two-ship, offensive mission against a composite force of fighters and bombers. The Demo 2 mission objective was to destroy the bombers while avoiding the escort fighters. The mission took place at high altitude.

The mission. The first incident highlighting the system's pilot-aiding capabilities occurred while still in the cruise phase of the mission. A stuck fuel valve required correction. Given prior authority, Pilot's Associate corrected the problem and advised the pilot. After crossing the forward edge of the battle area, the aircraft was advised of a target assignment to intercept the four-bomber strike force protected by four fighter aircraft. Pilot's Associate

created a new mission plan, including a probable intercept point consistent with premission attack strategies.

As threatening aircraft entered sensor range, Pilot's Associate allocated an on-board passive sensor to monitor the most critical threats. The system detected a combat patrol plane breaking orbit, but decided it was not an immediate threat to the mission. As the pilot flew the mission route, he could see his aircraft signature being managed with respect to known surface-to-air missile sites. This strategy



THE PILOT DID NOT NEED TO MANIPULATE DISPLAYS — PILOT'S ASSOCIATE COMPLETELY CONTROLLED DISPLAY SELECTION, CONTENT, PLACEMENT, AND FORMAT.

involves brief, expected detections by the missile sites, but minimizes exposure.

Pilot's Associate began suggesting minor course corrections to help the pilot maintain preferred intercept tactics, which involved remaining beyond visual range. As the missile launch point approached, the system detected a slight drop in engine oil pressure and presented a succinct message of minimal urgency to the pilot. The pilot had already authorized Pilot's Associate to activate the target tracking radar, which soon began to overheat. The pilot tracked the targets on his tactical situation displays, which provided missile launch acceptability regions. The system calculated and displayed offensive and defensive regions, and analyzed and suggested weapon allocations for both the lead plane and the wingman. The pilot launched missiles at the bombers while the oil pressure continued to drop and the radar reached a critical temperature. Although authorized, Pilot's Associate did not shut down the radar since it was still needed to illuminate targets.

The pilot was making his turn for egress and monitoring the missile flyout when an unknown surface-to-air missile launched.

Pilot's Associate advised the pilot of an effective defensive maneuver while it dispensed chaff and flares at appropriate intervals. The oil pressure problem became critical with imminent bearing seizure. The system had no authority over the engine and continued to keep the pilot updated. The evasive maneuver and countermeasures enabled the aircraft to survive the surface-to-air missile explosion; however, explosion debris damaged control surfaces and the remaining good engine. The system planned a new egress route to an alternate recovery base, which was consistent with the aircraft's remaining capabilities while minimizing exposure to further threats.

Technical overview. The Demo 2 system executed the 30-minute mission in approximately 180 minutes. This was roughly 1/6th real-time operation, although all measurements were made with development-level code and no debug or display code was removed or deactivated. We used a stop-clock approach to maintain time synchronization between the nonreal-time Pilot's Associate and the real-time aircraft simulation. When a subsystem needed to catch up with the simulation, it paused the simulation temporarily and effectively executed in zero mission time. To obtain an accurate picture of operating in a real-time environment, we recorded all message traffic between Pilot's Associate, the cockpit, and the simulation during a representative mission execution. Then, by adjusting the time tags of the messages that executed in zero mission time during simulation pauses, and replaying the messages through the simulation, we were able to model actual system execution in real time. The only liberty we took was adjusting message timing; we retained all errors, the most noticeable being an incorrect message for a system failure. Given this opportunity to view Pilot's Associate in real time via replay, the importance of display and interface formats became apparent.

The plan-and-goal graph and dictionary form the underlying functional skeleton for Pilot's Associate. The graph's hierarchical structure produces a common planning language among modules, and the dictionary serves as the explanation of that structure, its meaning, and its purpose. The need to communicate similar information about plans and goals to multiple subsystems, often in different formats, drove

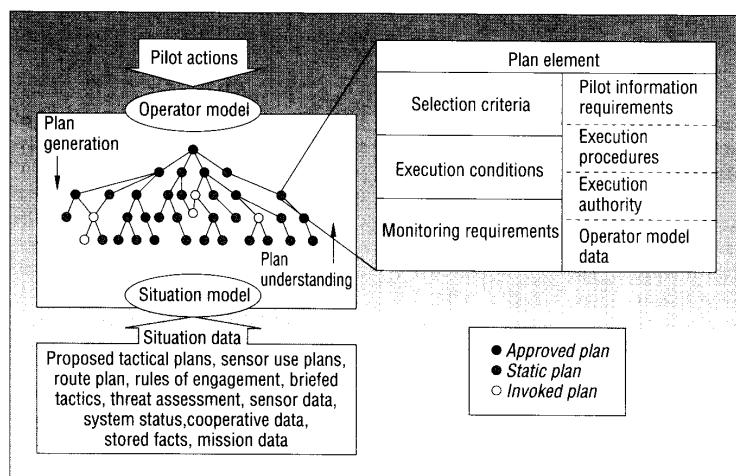


Figure 1. The plan-and-goal graph.

the effort to establish a common language. A representation of the graph and how it provides subsystem integration is shown in Figure 1. The graph is a relatively simple structure, resembling a conventional And/Or tree. A plan can be accomplished only by successfully completing all of its children goals; conversely, a goal can be satisfied by one or more of its children plans. The graph structure simply represents the dictionary, which contains detailed descriptions of each node, sufficient for uniform interpretation and proper subsystem implementation. Demo 2's graph and dictionary required configuration management, and proved difficult to maintain without automated tools.

The team structured this software in a heterogeneous, loosely coupled system. Individual subsystems were not restricted to a particular development environment or software approach; therefore, each individual subsystem was implemented with an approach considered appropriate for its domain. A sixth subsystem, the Mission Manager, originally served as a common communications link between these disparate software structures, and functioned as an invaluable integration tool for this loosely coupled system. The Mission Manager later maintained the global blackboard, the central repository for active plans and goals instantiated from the graph.

The Pilot-Vehicle Interface. The creation of an associate must be driven by requirements to support human decision-making capabilities, in this case through an

intelligent and intuitive interface. Therefore, the interface must be the central Pilot's Associate subsystem.

The Pilot-Vehicle Interface combines interface management, an adaptive aider, and an error monitor. This interface must be able to recognize pilot intentions and model human resources and performance. Because many plans compete for a pilot's attention, interface management selects the form, content, modality, and placement of information in the cockpit. For the Demo 2 cockpit, interface management allocated information to seven logical display surfaces on three physical displays and a voice synthesis channel. Interface management arbitrates between competing plans by considering importance and urgency, pilot resource demands as a result of presented information, the current system context established by the other subsystems, and the intent recognizer. The intent recognizer, a script-based reasoner, tries to explain pilot actions with respect to competing hypotheses instantiated from the graph's leaf nodes. Actions that the intent recognizer cannot explain are identified to the error monitor for consequence analysis and possible remediation. At Demo 2, the interface subsystem was implemented in Common Lisp and contained almost 2 megabytes of knowledge.

The most difficult aspect of demonstrating an intelligent and intuitive interface was that, as the interface became more successful, it became less apparent. Its features were difficult to detect unless they failed to function. After considerable work,

Demo 2 successfully showcased an intelligent cockpit where the pilot did not need to manipulate displays during missions — Pilot's Associate completely controlled display selection, content, placement, and format. No pilots criticized the system's choices or timing of displays. Most criticism centered on aspects of displays that the pilot could tailor, such as a north-up orientation versus track-up. Before the mission, pilots customized many aspects of the interface as well as levels of automation. The pilot could preapprove plans for execution, implicitly or explicitly approve or disapprove plans by actions, or ignore plans. When a plan was ignored, Pilot's Associate continued to be responsive to the situation and the pilot's needs. In all situations, the interface neither assumed authority nor negated the pilot's authority.

Situation Assessment. Demo 2's Situation Assessment subsystem connected the pilot and Pilot's Associate to the outside world. This subsystem maintained a database of air and ground objects using information from internal sensors (correlated by a sensor data manager external to Pilot's Associate) and from external sources data-linked to the aircraft. Situation Assessment inferred or calculated additional attributes for these objects, such as lethality, intent, priority, and impact. The subsystem maintained this track information and continually performed assessments for more than 40 objects in the environment: friendly, foe, and neutral. It managed uncertainty in threat intent using General Electric's Reasoning with Uncertainty Module.¹ RUM could also be used for dealing with uncertain or missing information; however, for Demo 2, perfect sensor information was assumed. For Situation Assessment code development, we used Intellicorp's Knowledge Engineering Environment² primarily because it supported RUM.

Two important aspects of the Situation Assessment design were information streams and monitors, which used information from the planner subsystems to focus assessment activities on tactical situations relevant to the other subsystems. Information streams were derived from the information requirements that other subsystems, primarily the Pilot-Vehicle Interface, imposed on Situation Assessment — that is, periodically collected or exception-based details about objects. The second form of

intermodule communication, the monitor, detected events affecting a plan and notified the requester. Monitors typically contained much less information than information streams and were often used to trigger an information stream for detailed communications. The combination of associating information requirements with current system plans and using monitors and information streams to meet those requirements allowed Situation Assessment to perform its functions while minimizing both computing time and message traffic.

Mission Planner. The Mission Planner's foundation was a route planner designed as a stand-alone, interactive system for strategic applications. Using a dynamic programming algorithm, the route planner incorporated complex aspects of route planning, such as aircraft signature and performance characteristics. As the design developed, this ground-based, point-to-point planner requiring a human operator evolved into an automated, on-board, point-to-region planner. The point-to-region capability essentially required running the dynamic programming algorithm in reverse. The goal node for the algorithm became the starting point, and heuristics were used to select an intercept route from the resulting paths. Control and evaluation functions, previously performed by a human operator, were implemented in Lisp as functions of the knowledge-based portion of the Mission Planner. This executive module used the route planner, implemented in Fortran, as a planning resource. Though capable of planning over a three-dimensional space, the route planner was limited to two dimensions for efficiency during Demo 2.

Tactics Planner. This subsystem reasoned about threats and targets for attack or evasion, ordered Situation Assessment to monitor high-interest threats, and directed sensor configurations to obtain needed information. It did not invent tactics, but recommended ones consistent with the pilot's own prebriefed tactics for the specific mission. However, suggested tactics were situation dependent, and the subsystem adapted the prebriefed tactics in areas of geometry and timing to fit the current, dynamic situation. Also part of the Demo 2 subsystem was a limited sensor-management capability. For example, the

Tactics Planner could select the most tactically critical air threats for continual passive monitoring.

The Tactics Planner was organized as a hierarchical tree of plan elements. Each element represented a small tactics subproblem and could be considered a planner capable of reasoning about its particular domain. This representation depended on the system's plan-and-goal graph and required a unique development environment. Kadet, a Lisp-based planning tool and reasoning engine built by ISX Corporation,

THE STRUCTURE REMAINED LOOSELY COUPLED AND HETEROGENEOUS. AS SYSTEM FUNCTIONALITY GREW, A COMMON LANGUAGE FOR SYSTEM PLANS AND GOALS PROVED NECESSARY.

formed the subsystem's foundation. The reasoning engine was a complex and specialized mechanism with access to blackboards, objects, plan elements, and facts. Each plan element contained knowledge expressed in rules. The Kadet rule system supported both forward-chained and backward-chained reasoning. To overcome some performance limitations associated with available AI development environments and tools, Kadet constrained reasoning within partitions of the overall knowledge base to improve response times.

System Status. This subsystem focused on assessing aircraft systems and on corrective procedures following fault diagnosis. The primary means to detect faults were parametric limit checks and interpretation of built-in test codes. A main focus was engine status, primarily due to readily available domain knowledge. The subsystem also evaluated proposed mission and tactical plans relative to aircraft performance limitations, and calculated the fuel flow matrix used by the Mission Planner to generate routes. At Demo 2,

System Status displayed architectural breadth and the ability to support future enhancements, but it severely lacked functional depth. It was originally implemented in KEE, but moved to Lisp code in the form of decision trees for efficiency.

Phase 1 culmination: Demo 3

Taking advantage of Demo 2 system evaluations and lessons learned, the Lockheed/Air Force team wanted to make changes in software design, implementation strategies, and development environment tools. Demo 3 was required to increase the current system's functionality and to focus on high-payoff requirements for a current or planned service platform. After choosing a candidate platform, we needed to shift from experimental computing architectures to those being developed for military application in the 1990s. The redirected program also needed to retarget Pilot's Associate functionality for specific mission/vehicle requirements and to identify a methodology to move system hardware and software into the selected aircraft. However, the original program philosophy — to apply complex, knowledge-based systems to aid pilots — survived.

In November 1989 we conducted Demo 3, the final stage of Phase 1. The selected service platform for the air-to-air effort was, naturally, an advanced tactical fighter (ATF is a generic term for the Air Force's "next-generation" fighter aircraft). As in Demo 2, this demonstration included a real-time mission replay and a live Pilot's Associate system running in nonreal-time. We also demonstrated results of real-time investigations and a skeletal implementation of the cooperating, real-time Tactics Planner and Situation Assessment subsystems.

The scenario. The Demo 3 mission scenario was a high-altitude, force-protection fighter sweep for a flight of strike aircraft. Four aircraft, named Knight Flight, were equipped with Pilot's Associate systems and operated as two two-ship elements (called ownship and wingman). Their mission objective was to protect eight F-15E strike aircraft (called Hammer Flight) from enemy fighters. Two additional fighters, called Cannon Flight and also equipped with Pilot's Associate, were given the task

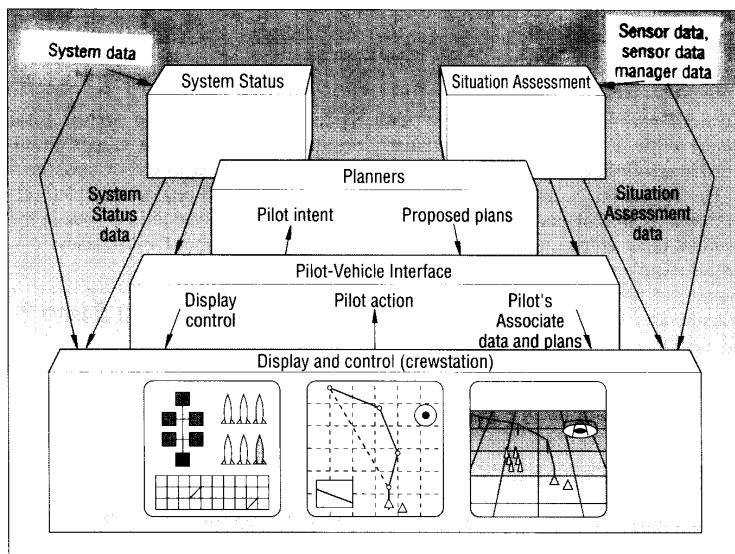


Figure 2. Dataflow in the Demo 3 Pilot's Associate.

of keeping the target area clear of enemy fighters immediately before and during the attack. The mission scenario included engagements beyond visual range.

The mission. The first visible Pilot's Associate action occurred prior to crossing the forward edge of the battle area. Pilot's Associate detected a fuel transfer failure and determined that the problem was a stuck fuel valve. Pilot's Associate toggled the fuel valve based on premission authorization from the pilot, and informed the pilot of the corrective action. Shortly afterward, a generator malfunctioned. Pilot's Associate isolated the fault and presented the recommended corrective action to the pilot. The pilot accepted the recommendation, the system performed the emergency checklist, the generator reset, and Pilot's Associate updated the pilot on the generator status. From this point on, using premission authentication information and code inputs, Pilot's Associate responded to friendly external requests for authentication and for friend-or-foe identification.

Immediately after crossing into the battle area, Pilot's Associate detected and assessed the mission impact of an unsuppressed surface-to-air-missile. Pilot's Associate generated a plan for evasion and recommended to the pilot a tactic to defeat the inbound missile. With the pilot's pre-mission approval, Pilot's Associate

activated appropriate countermeasures to help defeat the missile. Then Pilot's Associate provided a low-observable route to rejoin the strike force and coordinated a two-ship weave maneuver with the wingman to match the speed of the strike flight.

Pilot's Associate continually monitored the changing environment. By using geometrical information (position, heading, altitude, speed, etc.) regarding its own aircraft, the wingman, and other cooperating elements, Pilot's Associate evaluated the best use of all aircraft resources and executed sensor plan contracts to provide the most current and vital information about threatening objects in the environment. Pilot's Associate detected an enemy combat air patrol breaking orbit, and interpreted the action as hostile intent directed at the strike flight. The system then detected a second element of enemy aircraft and determined that it also posed a threat to the strike flight. Pilot's Associate sorted and assigned the threats to Knight Flight members according to the lead pilot's premission targeting contract. The system helped the pilot with the suggested tactical plan by coordinating the maneuver geometry, sensor usage, weapon selection, and launch opportunity. As Knight Flight maneuvered to launch, a hostile aircraft launched an air-to-air missile toward the aircraft, but the pilot evaded the missile with a maneuver suggested by Pilot's Associate. While

successfully engaging and defeating the threats, Knight Flight became separated from the strike flight. Pilot's Associate planned a rejoin route while continuing to support the need to minimize exposure and maintain mission constraints. Knight Flight rejoined the strike flight and, due to a mission abort of Cannon Flight, swept the target area. With a successful sweep, Knight Flight kept the target area clear of enemy fighters while the strike flight attacked the target and accomplished the primary mission objective.

As more threats were encountered, Pilot's Associate assessed the danger, notified the pilot of a surface-to-air missile launch, and provided an optimum evasive maneuver. A missile explosion damaged the right engine and right leading-edge flap. Pilot's Associate assessed the damage and used the degraded flight envelope and reduced flight-control response to plan recovery. Pilot's Associate notified the pilot of the damage and provided appropriate checklists to deal with this multiple emergency. The system planned an egress route, taking into account the aircraft's degraded performance capability and the need to land at the nearest suitable base once in friendly territory. It presented the pilot with the egress route and suggested primary and alternate landing bases suitable for the emergency situation.

On the egress, the inertial navigation system and a generator failed. Pilot's Associate notified the pilot, who called for the generator checklist to see which subsystems were affected. The pilot completed the checklist provided by Pilot's Associate and reset the generator. Pilot's Associate helped realign the inertial navigation system so that it again functioned properly. Once in friendly territory, Pilot's Associate provided information on the emergency landing base: current base status, weather, runway parameters, and approach procedures. The system also helped the pilot file an in-flight report of the successful mission.

Technical overview. The Demo 3 Pilot's Associate performed better than the Demo 2 system, with a tremendous increase in functionality. The more complex fighter-escort mission executed at roughly three times real time (with development-level code, debug code, and display code removed or deactivated). Demo 3 used the

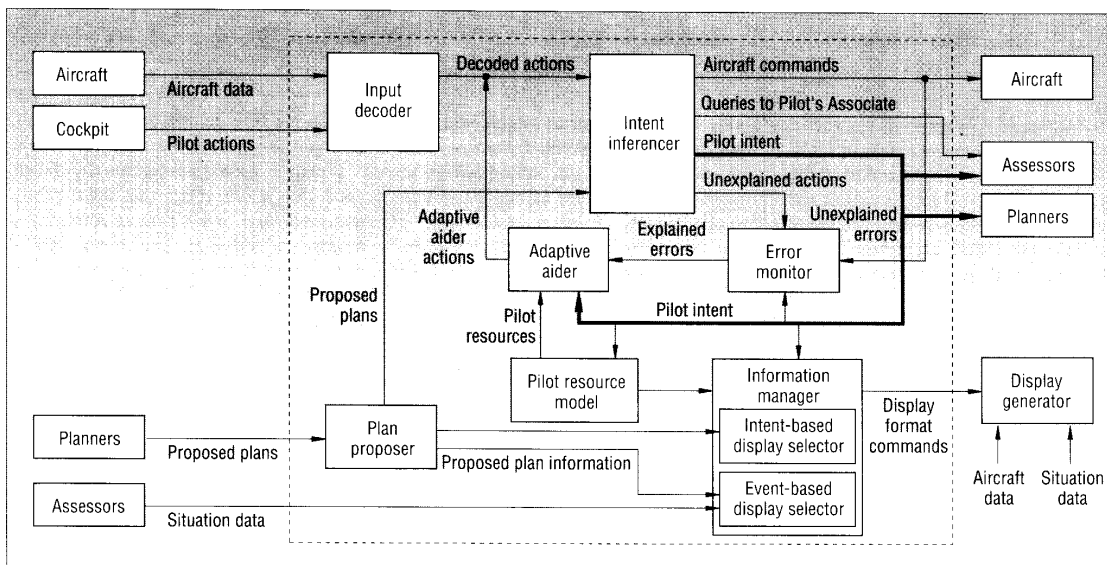


Figure 3. The functional design of the Demo 3 Pilot-Vehicle Interface.

same stop-clock approach as Demo 2 to stay in step with the real-time simulation.

The basic structure of this approach remained a heterogeneous, loosely coupled system, with each subsystem implemented in a domain-appropriate fashion. The Demo 3 concept of system dataflow is depicted in Figure 2. The Mission Manager subsystem remained as the keeper of the global blackboard, a repository for active plans and goals, and a communications channel between subsystems. However, as Pilot's Associate matured from Demo 2, many of the early functions of Mission Manager were decentralized into the individual subsystems.

The plan-and-goal graph and dictionary remained, increasing in importance. As system functionality grew, a common language dealing with system plans and goals proved necessary. However, updating graph information became cumbersome as development continued and as the functionality represented by the graph increased. The need to continue the graph concept led to the development of automated tools to maintain and update its structure and dictionary.

At a more detailed level, the Demo 3 system featured several new functions and improvements to the cockpit, displays, and simulation environment. It also incorporated many reactive elements, realistic sensor models, and planning considerations of

additional flight elements and an escorted strike group. The pilot could now query the system for information on navigation, threats, and systems. The Demo 3 system could plan interactively with the pilot and be more responsive to pilot inputs.

A key feature of pilot interaction and confidence in system performance was the Mission Support Tool, which originated as an idea of the program's Operational Task Force, a program advisory group whose members had considerable tactical fighter experience. This tool demonstrated how the pilot could set authorization limits for Pilot's Associate and tailor its planning performance regarding engagement tactics, the use of sensors and countermeasures, mission planning, communications, and electronic warfare. The Mission Support Tool was not a new idea; pilot tailoring of the system existed at Demo 2. However, with increased system functionality, particularly in tactics planning, responsiveness to pilot needs and desires required a more extensive tool for pilot tailoring. For the first time, the complete vision of Pilot's Associate as a true partner for the pilot emerged.

The enhanced performance of Pilot's Associate resulted from integrating the full Demo 2 system capability with functionalities developed for specific Demo 3 requirements. We could not possibly describe adequately all of the system's capabilities;

the following subsystem accomplishments are simply highlights.

The Pilot-Vehicle Interface. This subsystem retained Demo 2's integrated design, but its individual functions evolved into new implementations. The design approach was to try to reduce pilot overload through

- intent inferencing — inferring pilot intent and communicating it to the other subsystems;
- display management — configuring displays and controls according to pilot information requirements and intelligently communicating messages to the pilot; and
- adaptive aiding — performing preapproved tasks, detecting possible pilot errors, determining their consequences, and proposing error remediation if necessary.

All Demo 2 interface functions were expanded for Demo 3, as shown in Figure 3. One significant enhancement, pilot intent recognition, was accomplished through three types of input. In addition to the script-based reasoner developed for Demo 2, Demo 3 added a plan-based reasoner, which tried to explain pilot actions based on plans and goals, as well as situational data about the state of the world. The plan-based reasoner was a more robust effort to infer pilot intent, allowing the subsystem

to recognize pilot intent when pilot actions were not part of a current task.

Two additional concepts complemented the original design. First, interface management evolved into an intent-based display selector and an event-based display selector. The intent-based selector encompassed the original design for this function, configuring cockpit displays with the most pertinent information derived from the active plans inferred by the intent recognizer as well as from plans proposed by other subsystems. The event-based selector determined the best method and modality for telling the pilot about a specific event. Each event type had an "expert" that decided the method of pilot notification. Because the expert reasoned about various display resources, the system was better able to deliver a significant event message to the pilot.

The second new concept implemented, the plan proposer, received plans from other subsystems and intelligently presented these recommendations to the pilot.

We continued to work on the subsystem in Common Lisp. At Demo 3, the subsystem contained 6.2 megabytes of total source code, and its knowledge base contained 3.4 megabytes of knowledge. The knowledge not contained explicitly in the

plan-and-goal graph was expressed using 828 rules. Execution times were much improved from Demo 2, falling within the response time goal for all major functions except the interface management areas of the intent-based selector and the event-based selector. Therefore, the interface subsystem was in an excellent position for the major focus of Phase 2 development — achieving real time.

Situation Assessment. This subsystem achieved major functionality leaps from Demo 2 and also underwent design revisions that maintained performance while increasing capability (see Figure 4). One major improvement was the inclusion of "real" data into the system. The subsystem no longer reasoned with perfect data and knowledge about everything in the environment. Developers implemented an interface with a generic sensor data manager, in which sensor models portrayed generic, but realistic, real-world sensors. The Situation Assessment subsystem developed a structure to handle four types of object data uncertainty: uncertainty in measurement error, uncertainty in attributes with discrete values (such as class or type), uncertainty representing the probability that an object actually exists, and uncertainty due

to missing information at the numerical, discrete, or object level.

Situation Assessment gradually developed into a subsystem that intelligently managed how and when to perform its assessment activities. This was a significant improvement, increasing both functionality and performance. We used KEE initially to develop the subsystem because of its compatibility with the RUM development environment. After we finished Demo 2, RUM-Runner became available, which let us compile the RUM rules. Using compiled rules eliminated the need for KEE; therefore, we rewrote the subsystem using Lisp to store data and the Automated Reasoning Tool to control the monitors and information streams. In addition, we hierarchically ordered the conditions of each monitor evaluation so that the least computationally expensive conditions for monitor success were checked first. These levels were designed to increase monitor efficiency while decreasing computation time.

Approaching Demo 3, the Situation Assessment software design was stable, and efficiency dictated removing all development tools from the subsystem software and rewriting the control portion of the subsystem in Lisp. Plan-based assessment

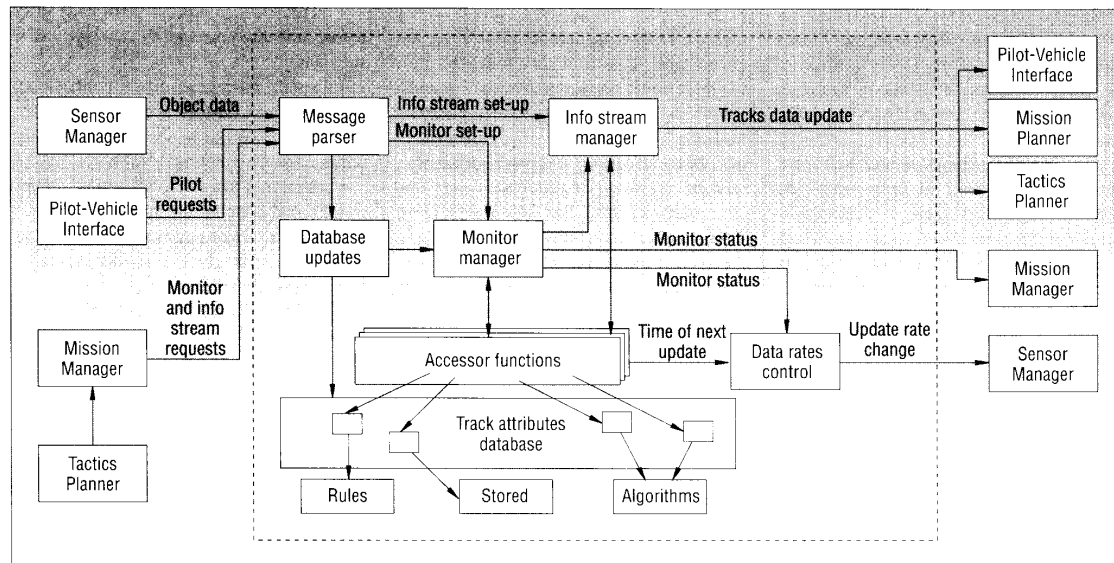


Figure 4. The functional design of the Demo 3 Situation Assessment subsystem.

in the Demo 3 system also increased the subsystem's efficiency by using monitors and information streams to focus assessment. The development of Situation Assessment through Phase 1 completed a cycle from an overworked, number-crunching program to a subsystem that intelligently controlled assessment and computation, and disseminated that information in a computationally efficient manner. At Demo 3, final Situation Assessment code included 51 rules for managing processor resources, 150 RUM rules for calculating uncertain parameters, 37 processing functions, and more than 0.7 megabytes of Lisp source code.

Mission Planner. This subsystem underwent major revisions after Demo 2, as Figure 5 shows. However, the original functions remained: an executive for heuristic control, assessment, and evaluation, and a dynamic-programming algorithm for route planning. The original route planner, hosted on a VAX 11/780 along with the Pilot's Associate aircraft-simulation software, presented problems as implemented for Demo 2. The computer resource contention between the simulation and the dynamic-programming route planner caused extremely slow execution. Although the executive

reduced the search space and runtime by using a heuristic search algorithm to find subplanning theaters as a preprocess to the algorithm, this approach was still slow and inefficient.

We redesigned the route planner for Demo 3 to improve its application in the airborne planning domain, implementing the planner in C for software efficiency, and hosting it on a Sun workstation to increase computational efficiency. This route planner was no longer limited to a two-dimensional planning space. It represented routes in three dimensions: longitude, latitude, and altitude. However, no time dimension was implemented. A time-based planning approach would provide solutions to problems regarding moving air threats, positional relationships to escorted flights, and multivelocity requirements that were difficult or impossible to handle with the Phase 1 approach. For Phase 2, we plan to consider time-based planning as well as specialized route-planning hardware to increase the Mission Planner's functionality and to decrease runtime.

The heuristically controlled Mission Planner executive used the route planner to supply routes from which to select. The executive examined different routes and evaluated costs and options before the

subsystem made a selection or formulated a mission plan. The Demo 3 executive could handle both the Demo 2 intercept mission and the Demo 3 escort mission, each requiring unique heuristic control. An intercept mission required that the routes be evaluated for speed, fuel, timeliness, and threat detection. An escort mission needed a route based on speed, fuel, time to return to the original course, average distance from the escorted package, and threat detection. After a mission replan triggered by an off-course detection, new threat data, or an aircraft performance change, the Mission Planner executive assessed the mission type and the current context to determine which route phases to generate and what goals to achieve. This unique heuristic control and the ability to evaluate a new mission plan in terms of an escort mission were major accomplishments for Demo 3. Despite the increase in this subsystem's functionality and complexity, its runtime performance improved by a factor of 20, putting Phase 2 real-time execution well within reach. At Demo 3, the Mission Planner executive contained 150 rules for data management and 0.6 megabytes of Lisp source code.

Tactics Planner. Through the global blackboard, the Tactics Planner communicated

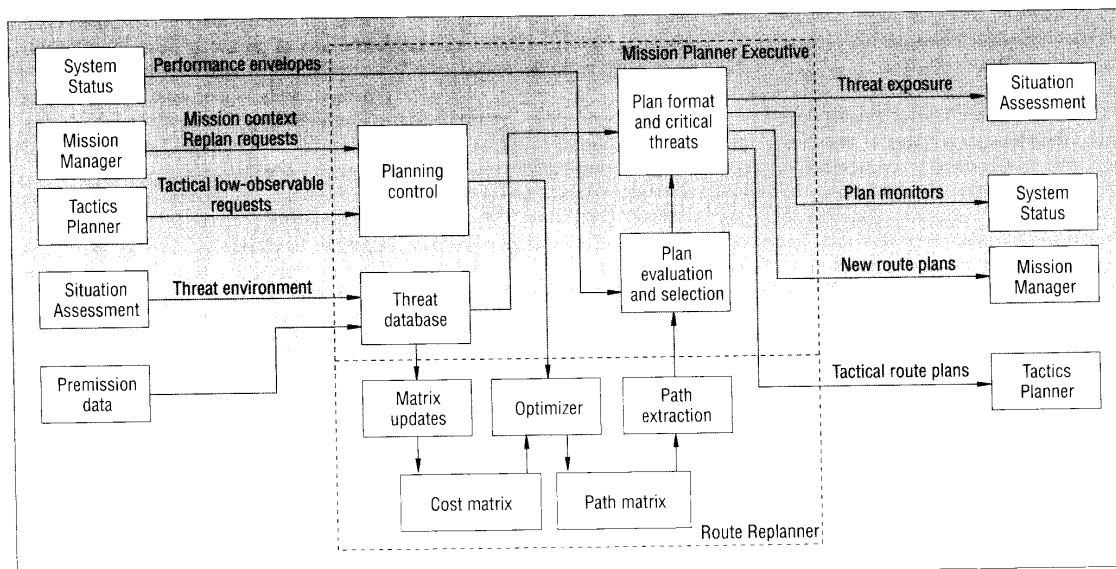


Figure 5. The functional design of the Demo 3 Mission Planner.

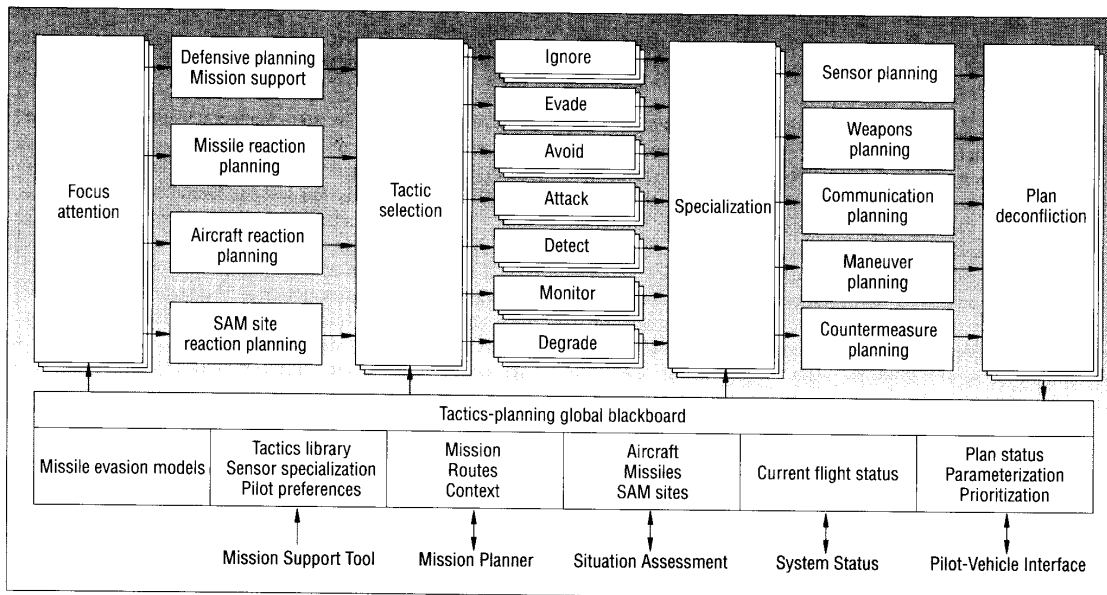


Figure 6. The functional design of the Demo 3 Tactics Planner.

with each Pilot's Associate premission and resident subsystem. It communicated with

- the Mission Support Tool to provide mission- and pilot-specific preferences,
- the Mission Planner for mission routes and context,
- Situation Assessment for track object assessment information,
- System Status for current flight status, and
- the Pilot-Vehicle Interface for plan status and pilot-plan interaction.

Conceptually, the Tactics Planner comprised four functions: focus of attention, tactic selection, specialization, and deconfliction (see Figure 6). The focus-of-attention function allocated planning resources to the most pressing needs. The subsystem evaluated many alternatives to support these high-priority activities and made choices among multiple types of offensive and defensive tactics. Once an approach had been identified, a specialization function decomposed the tactic to its lowest level. This lowest level focused on the basic aircraft attributes that the Tactics Planner could actually affect, such as the use of sensors, weapons, communications equipment, and countermeasures. During plan specialization, each resource was evaluated with respect to its own aircraft and to the wingman to provide an effective use of

all resources. Finally, the Tactics Planner performed a plan deconfliction function because, in the course of conducting many types of planning, the subsystem generated plans as solutions to one goal that could conflict with plans generated as solutions to other goals. This final process of plan prioritization and deconfliction assured that Pilot's Associate would suggest achievable plans.

To support the pilot, the Demo 3 Tactics Planner reasoned in the following areas:

- Mission support planning (formation planning, mission phase support).
- Communication support (radios, data link, identifying friends and foes).
- Coordinated offensive and defensive sensor planning (searching, monitoring, weapons support).
- Surface-to-air missile site reasoning (searching, monitoring, reaction planning).
- Enemy missile reasoning (searching, monitoring, evasion planning).
- Countermeasures and expendables planning.
- Air-to-air engagement planning (support when within visual range, evading when beyond visual range, avoiding, degrading, ignoring, and attacking).
- Weapons support.
- Signature management (radar and infrared).

An area of significant emphasis for Demo 3 development was the attack subsystem for planning air-to-air engagement when beyond visual range. Attack planning coordinated the use of weapons, sensors, maneuvers, and communications between the lead and wingman Pilot's Associate aircraft and, to a lesser degree, among additional cooperating aircraft. The use and capabilities of realistic sensors, and the use of weapons in attack planning, required significant knowledge acquisition to accurately support actual pilot information-gathering techniques and the deployment, use, and support of advanced missile systems and tactics.

We continued to express Tactics Planner knowledge in the form of Kadet rules within the planning framework of the plan-and-goal graph. The software contained 379 rules partitioned into 148 rule sets, 208 planning elements, and 219 functions. The Demo 3 implementation required 1.6 mega-bytes of Lisp source code. While the knowledge base occupied more than 1 megabyte, it was composed primarily of rule macros that expanded to several times their original size before they were compiled to object code. Developers estimated that the Demo 3 knowledge base contained only 40 percent of the knowledge required for a fully functional Phase 2 Tactics Planner.

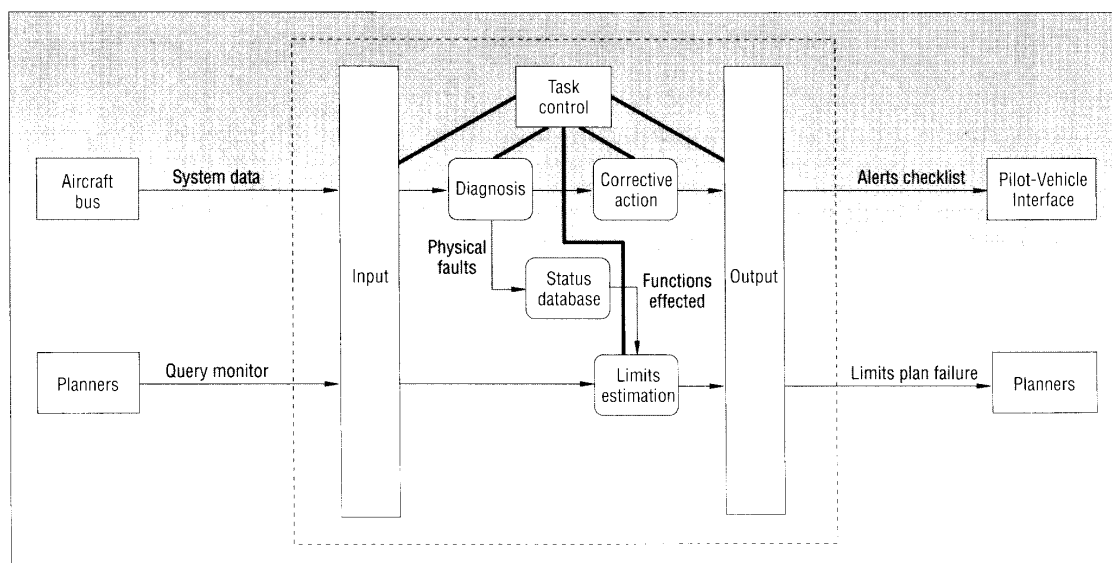


Figure 7. The functional design of the Demo 3 System Status subsystem.

System Status. As shown in Figure 7, this subsystem's functional development continued in aircraft diagnosis, limits estimation, and corrective action, matching the extensive architectural structure implemented in Demo 2. This stage focused primarily on diagnosis, including model-based reasoning, built-in tests, and limit checks. The diagnosis function monitored the aircraft control and communications bus to detect and isolate faults, and communicated these results to the subsystem's corrective-action and limit-estimation functions. We focused on the engine subsystem because detailed models of both normal and abnormal engine behavior were available for analysis.

The limit-estimation function reported operating constraints to the planner subsystems to ensure the development of feasible plans. This function also monitored these plans for continued feasibility during execution. The Demo 3 subsystem provided the following classes of information:

- General limits on maneuvers such as the maxima for altitude, turn rate, and climb angle.
- Caution and warning status for major systems such as weapons, engines, and electrical power.
- Status of consumables such as fuel, flares, and missiles.

- Specific limits in the form of a fuel-flow matrix, used by the Mission Planner for setting the fuel budget and mission profile.

- Specific limits on maneuvers to guide the Tactics Planner in selecting feasible maneuvers.

The corrective-action function generated options for action by the pilot or by the Pilot-Vehicle Interface when emergencies occurred because of equipment failure or battle damage. Corrective actions for Demo 3 included procedures to correct a stuck fuel valve, a generator failure, and a failure of the internal navigation system.

Because Demo 2 concentrated on building an architecture for a fully developed System Status subsystem, this implementation did not change in Demo 3. It contained 260 rules to manage control flow and 1.1 megabytes of Lisp source code. However, using decision trees to implement corrective action functionality did not work: They could not handle multiple faults or track procedure execution through mission context changes as well as developers had expected. The explicit representation of possible combinations was too large to be practical. A future solution might be to represent action elements that can be assembled in appropriate combinations at runtime, much in the way that the

Tactics Planner created tactical plans. System Status development for Phase 2 will focus on capabilities assessment and limit estimation due to the information requirements of other Pilot's Associate subsystems.

Real-time investigation efforts. As the program progressed through Phase 1, we added real-time analysis as a risk reduction measure. We examined real-time performance bottlenecks and investigated and demonstrated hardware and software architectures or techniques to address these bottlenecks. Our goal was to demonstrate a clear, reachable path to achieve real-time performance in Phase 2, while using the computing capacity expected to be available in a generic advanced tactical fighter.

We identified five tasks to support real-time analysis:

- (1) Produce an integrated development environment to support the development and testing of real-time models.
- (2) Develop a runtime environment designed for efficiency with few, if any, user-interface or development features.
- (3) Provide a simulation harness that lets us estimate the behavior and performance of application software running on a target machine, given the host code on a host machine, and given data comparing target and host software performance.

(4) Develop analysis tools that let users record, observe, and analyze the behavior of application software.


(5) Demonstrate the development and runtime environments.

For the integrated development environment, we used the ABE/RT real-time tool,³ which is part of the Cimflex-Teknowledge ABE/RT project sponsored by DARPA's Strategic Computing Program. ABE/RT let us develop a real-time Pilot's Associate model, scheduling facilities, and a remote execution capability. We also specified a C-based runtime environment for the Sun workstation to run the software produced in the ABE/RT environment. The components of the runtime environment were an executive and a code generator. The executive provided necessary runtime support for the real-time system, while the code generator took as input a compiled ABE/RT system and produced a set of C++ files that defined all necessary system and external interactions. The user could then fill in the skeletal C++ code for the particular component.

The simulation harness and analysis tools were crucial efforts in determining the performance of Pilot's Associate software for real-time implementation. The basic simulation harness let us track and modify the clocks used by several system activities, and it let us externally manipulate those clocks to provide roughly synchronized execution of processes running at different natural rates. We extended this basic harness to work on a host system distributed across multiple Symbolics machines and Sun workstations to allow a multimachine system simulation. System designers could then simulate software operation on different architectures, model various processor execution speeds and multiprocessor configurations, and compare the resulting execution characteristics. The analysis tools consisted of logging facilities and a logic analyzer. The logging facilities let the user keep a record of the start and end of each subsystem activation across multiple machines or within one machine. To view the logs, a logic analyzer plotted time against channels of events of various types. The events, displayed in a strip chart format, let us study detailed interactions between software components. We also analyzed the data statistically to determine the minimum, mean, and maximum execution times of

various tasks and their relationship to previously specified deadlines.

We held two separate real-time demonstrations in conjunction with Demo 3. The first was a depth-first approach of actual real-time performance. The objective was to validate the simulation harness and scheduler by comparing predicted behavior with actual measurements. We made the measurements using real-time versions of the Tactics Planner and Situation Assessment subsystems, developed and running in the ABE/RT environment. Dummy versions



THE ACTUAL IMPLEMENTATION OF A SYSTEM OF THIS COMPLEXITY UNCOVERS MANY GAPS IN TECHNOLOGY STILL TO BE ADDRESSED BY THE RESEARCH COMMUNITY.

of other Pilot's Associate tasks provided realistic system loading. Data logging and analysis features were also illustrated.

The second demonstration was a breadth-first approach, containing the tools and techniques necessary to predict and guarantee performance of the total system, and showing the current computational model of Pilot's Associate. The objective of this demonstration was to show the scalability and usefulness of the development environment under full load.

This real-time development and demonstration work produced results in system design and task scheduling. Real-time analysis led to the recommendation that the baseline scheduling paradigm should be an event-driven, asynchronous system of tasks in which the clock interrupts are treated like any other event. As a result of real-time testing, we also decided to make as many task-scheduling determinations as possible off-line at design time. The off-line determinations included task allocation to a specific processor, scheduling modes of operation, task priority within a mode, and computational resources assigned to each active process in a mode. These

decisions, in conjunction with the development environment, analysis tools, and demonstrations, laid a convincing foundation for a real-time Pilot's Associate.

What worked

Several issues contributed to the project's success.

System development. The software development environment available with symbolic processors increased productivity significantly, and greatly enhanced software prototyping, testing, and debugging. Ada developers need tools of this type before they can efficiently develop software of this complexity.

Rapid prototyping proved an effective methodology in developing Pilot's Associate functionality. Early prototypes provided working models for user evaluation and a baseline for requirements development for later prototypes. Major design deficiencies could be identified early to prevent major explorations of nonsensical paths. The ability of the incremental prototypes to demonstrate capabilities and interim successes proved important in terms of developer morale and user interest.

Involving the technical community. A technology advisory board composed of leading AI researchers in industry and academia provided a positive influence on system development. The board regularly reviewed major elements of a subsystem or system architecture. By defending developing ideas, we minimized risks and identified potential problems. The board also helped us consider the balance between success and risk.

Involving the user community. The Operational Task Force advised system developers in operational areas. Its participants had substantial flight crew and engineering backgrounds, and were selected based on their tactical fighter experience, understanding of Pilot's Associate technical issues and approaches, and availability for knowledge acquisition and product evaluation throughout the program. The group played a role in all phases of program development: requirements definition, design, development, and testing. This group not only had a role in functional

development, but also advocated Pilot's Associate concepts to the operational world and served as a valuable interface to the rest of the user community. These individuals provided input to the program from the user community and helped to ensure the realism and user acceptance of Pilot's Associate. We cannot measure the value of this group to realistic program goals and user community acceptance.

THE TECHNOLOGY OF PILOT'S Associate can be applied across a broad spectrum of applications. In particular, real-time, interactive process control applications are likely candidates. Pilot's Associate technology is being applied to helicopters, multicrew aircraft, submarines, and even unmanned vehicles.

In accordance with the program goal to provide a pull on the technology base, the program often needed nonexistent technology to implement the design. Uncertainty-reasoning approaches, pilot-modeling techniques, and development tools are areas that lacked sufficient research to implement the Pilot's Associate vision. Verification and validation of knowledge-based systems constitute another area that is lacking techniques and is quickly becoming a program issue. The actual implementation of a system of this complexity uncovers many gaps in technology still to be addressed by the research community.

References

1. P. Bonissone, S. Gans, and K. Decker, "RUM: A Layered Architecture for Reasoning with Uncertainty," *Proc. DARPA Knowledge-Based Systems Workshop*, The American Institute of Aeronautics and Astronautics, Washington, D.C., 1987, pp. 123-131.
2. *KEE Software Development System—Core Reference Manual*, Intellicorp, Mountain View, Calif., 1986.
3. J. Lark et al., "Concepts, Methods, and Languages for Building Timely Intelligent Systems," *J. Real-Time Systems*, Vol. 2, No. 1/2, May 1990, pp. 127-148.

Further reading

C. Leavitt and D. Smith, "Integrated Dynamic Planning in the Pilot's Associate," *Proc. AIAA Guidance, Navigation, and Control Conf.*, The

American Institute of Aeronautics and Astronautics, Washington, D.C., 1989, pp. 327-331.

C. Lizza, "Pilot's Associate: A Perspective on Demonstration 2," *Proc. AIAA Computers in Aerospace Conf.*, The American Institute of Aeronautics and Astronautics, Washington, D.C., 1989, pp. 386-394.

B. Pomeroy and R. Irving, "A Blackboard Approach for Diagnosis in Pilot's Associate," *IEEE Expert*, Vol. 5, No. 4, 1990, pp. 39-46.

B. Pomeroy, H. Spang, and M. Dansch, "Event-Based Architecture for Diagnosis in Control-Advanced Systems," *Artificial Intelligence in Eng.*, Vol. 5, No. 4, 1990, pp. 174-181.

T. Whiffen, M. Broadwell, and D. Homoki, "Intelligent Control of Situation Assessment Systems," unpublished paper presented at the Fourth Annual Lockheed AI Symposium, Mar. 1989. Available from this article's authors.



Sheila B. Banks is an Air Force captain and chief engineer for the Pilot's Associate Program in the Joint Cockpit Programs Office of Wright Laboratory, located at the Wright-Patterson Air Force Base. She directs the design and development of artificial-

intelligence software to provide information management and decision support. Her other research interests include knowledge acquisition, knowledge base support, and intelligent-systems verification and validation. She received a BS in geology from the University of Miami and a BS in electrical engineering and an MS in electrical and computer engineering from North Carolina State University.



Carl S. Lizza is an Air Force major and program manager for the Pilot's Associate Program in Wright Laboratory's Joint Cockpit Programs Office. He directs research into the application of artificial intelligence and advanced computing technologies

to provide information management and decision support systems for future single-seat fighter aircraft. His research focuses on integrating AI, conventional algorithms, parallel processing, and symbolic computing into a real-time, man-in-the-loop simulation environment. He received his BS in computer and information science from Ohio State University in 1977 and his MS in computer science from Wright State University in 1984.

Readers can reach the authors through Captain Sheila Banks, Dept. of the Air Force, Headquarters Aeronautical Systems Division, Wright-Patterson Air Force Base, Ohio 45433-6553.

JUST-RELEASED BOOKS

NEAREST NEIGHBOR (NN) NORMS:

NEAREST NEIGHBOR PATTERN CLASSIFICATION TECHNIQUES

edited by Belur V. Dasarathy

This book covers the past four decades of research in the area of nearest neighbor (NN) techniques within the field of pattern recognition, though its emphasis is on the more recent studies. It contains results and conclusions on nearly 140 studies grouped into ten categories, each dealing with a specific aspect of development in NN pattern classification techniques. This tutorial begins with a comprehensive survey of the field that includes a detailed bibliography of all the studies covered. The text contains reprints of 52 studies selected from the larger set of studies explored in the survey chapter.

464 pp. 1991. Hardbound, ISBN 0-8186-8930-7.
Catalog # 1930 \$65.00 / \$45.00 Member

call 1-800-CS-BOOKS

FROM IEEE COMPUTER
SOCIETY PRESS

