15

AUTOCREW: A Paradigm for Intelligent Flight Control

Brenda L. Belkin
AT&T Bell Laboratories
Middletown, NJ 07748

Robert F. Stengel
Department of Mechanical
and Aerospace Engineering
Princeton University
Princeton, NJ 08544

Abstract

An expert system Pilot-Aid is envisioned to automate many functional and low-level decision-making tasks in future high-performance and jet transport aircraft to help alleviate pilot workload. This chapter presents a design methodology for the development of multiple cooperating rule-based systems for the aircraft domain. Nine modular rule-based systems, collectively called AUTOCREW, were designed to automate functions and decisions associated with a combat aircraft's subsystems. The nine AUTOCREW knowledge bases were designed individually; areas of cooperation between the knowledge bases were identified, and common information was designated as "shared" information. Software tools were developed to aid in high-level design of the cooperating ensemble. An interactive graphical simulation testbed was developed to demonstrate and test the cooperating AUTOCREW ensemble's performance. Workload metrics were formulated to quantify AUTOCREW's performance in terms of the ensemble's efforts in assisting the Pilot. The workload metrics give reasonable results for the comparison of workloads among AUTOCREW's experts, as well as comparative results among task groups within a single knowledge base. The applicability of the methods utilized to design AUTOCREW for other applications is also discussed.

INTRODUCTION

This chapter describes a general approach to designing and prototyping cooperating rule-based systems. The approach was applied to the high-level design of an expert system Pilot-Aid to automate many functional and low-level decision-making tasks for aircraft operations. The details of the expert system can be found in [1]; this chapter is intended to point out some of the critical issues associated with the design of cooperative rule-based systems.

Pilots must make quick decisions and perform a myriad of time-critical tasks throughout a flight. The pilot bases her decisions on currently available information as well as her experience and judgment. During some phases of a mission, the quantity of data and information available may be so great that the pilot becomes incapable of making timely decisions and performing critical tasks. Pilot workload studies show that the high-stress, high-workload flight environment results in reduced pilot performance as well as reduced mission effectiveness [2]. Studies undertaken for both commercial transport and combat operations suggest that on-board pilot assistance and selective task automation are highly desirable features for future aircraft system designs [3,4]. The expected increase in aircraft system complexity and corresponding increase in pilot workload has motivated investigations into the potential applications of artificial intelligence (AI) theory in the flight domain [5-9].

Expert systems, which are computer programs usually developed in a symbolic processing language such as LISP or PROLOG, have emerged to solve difficult domain-specific problems. The expert system designer extracts heuristics and specific knowledge from domain experts. This information is used to formulate a knowledge base consisting of parameters and rules. An inference engine uses rules to conduct a search and to set parameters, thereby inferring knowledge about the problem domain. Expert systems are usually self-contained and are well-suited to a specific problem area. They are usually designed to operate in a stand-alone environment.

Expert systems have been developed in the areas of navigation, emergency procedures, and air traffic control, to name a few [10-13]. The introduction of multiple system concepts such as global blackboard architectures for information exchange between knowledge bases has been successful in aerospace implementations [14]. However, methods in multiple knowledge-base development, rule-based systems integration, and ensemble prototyping need further research if complex systems are to be systematically developed for flight domain operations or other problem areas.

A logical task classification scheme is a crucial factor in the successful development of multiple rule-based systems. In this research a logical task structure for the combat aircraft domain was developed, using as the structural paradigm a World War II bomber crew [15]. The tasks each bomber crew member performed at each mission phase were clearly defined. Based on this model, an ensemble of nine cooperating rule-based systems called AUTOCREW [1] represented as pilot copilot, navigator, flight engineer, radio operator, spoofer, lookout, attack, and defender was developed. Each rule-based component figuratively emulates a crew member's task

responsibilities. The overwhelming advantage of this organization is that each component system is intuitively obvious in function to the end-user. This facilitates the integration of the system into human operations.

Since systems designers and pilots think in terms of specific tasks such as navigation, aircraft systems diagnosis, and flight control, the crew member approach provides modular rule-based system components. There are two main advantages to designing modular rule-based systems. The first is transferrability: The crew structure is applicable to all aircraft types. For a civil transport application, the combat-specific modules are removed from the ensemble. In addition, modularity facilitates initial design and debugging, and promotes software reuse and capability growth. This approach to developing a complex AI-based pilot aid differs from the approach taken in preliminary descriptions of the Pilot's Associate Program [3,8]. In the latter, tasks are organized into categories called "Managers". The five cooperating Managers envisioned for the Pilot's Associate are categorized as Mission, Tactical, Situation Awareness, System Status, and Pilot-Vehicle Interface Managers. These classifications are quite broad and require additional breakdown to make them amenable for realistic development and implementation.

A major challenge in the successful development of multiple rule-based systems is integrating individual knowledge bases. One of the problems encountered is that software capabilities for information exchange between the systems are usually nonexistent -- each system is designed to be self-contained. This necessitates redesigning each knowledge base to accommodate the sharing facilities (or "shared" parameters) that provide the information exchange capability. A better approach would be to define the ensemble requirements on a macroscopic level before designing and implementing the individual systems in detail. When functions are represented on a high level, making changes to the knowledge bases is easily accomplished. The identification of shared parameters is easier when less knowledge-base detail is present. Therefore, this alternate approach provides the designer with an overview of an integrated system. Once the designer is satisfied with the prototype functions, cooperation, and logic flow, each knowledge base can then be developed in detail.

The development of software tools and techniques that aid in ensemble design, prototyping, testing, and evaluation is an important factor in the design of multiple rule-based systems. In this research, search-effort metrics called rule and parameter fractions are defined to quantify and compare knowledge-base workload distribution among component systems. Search effort information can help identify workload situations for which additional computer resources are required. These metrics can also be used to help identify task groups for which separate knowledge bases should be developed. In keeping with the high-level development philosophy discussed above, the designer should be free to focus on the overall system requirements and important elements of the prototype design and be less involved with implementation details. Definition of the knowledge-base functions, logic, and inter-system cooperation areas is the designer's priority. Prototype development tools should assist the designer in accomplishing this goal. A simulation testbed was developed to visually demonstrate intersystem cooperation and pilot-system interaction. The simulation testbed also provides the vehicle for knowledge-base logic verification.

AUTOCREW DEFINITION

Nine multiple cooperating rule-based systems for the combat aircraft environment were developed and implemented in the AUTOCREW expert system. Each component rule-based system is based on a typical World War II bomber crew member having specific task responsibilities. The Pilot must identify with the task performed by the on-board aid, so a crew-model design of component knowledge bases is commensurate with the Pilot's experience and training. As crew members are modelled, flight personnel can readily assist in the design, implementation, and testing phases of complex knowledge bases. In most cases, the human expert can outline the details of task performance and can inject heuristics and insight into problem-solving logic. A natural task classification scheme is an important factor in the success of complex knowledge bases. The AUTOCREW task architecture is motivated by these issues.

The block diagram in Fig. 1 depicts the integration of AUTOCREW within a Pilot-vehicle framework. The AUTOCREW crew members are responsible for performing tasks and controlling functions associated with the aircraft and on-board systems. The modelled crew members are COPILOT (flight control, aircraft performance, Terrain Avoidance/Terrain Following (TA/TF)), ENGINEER (aircraft system diagnosis, reconfiguration, emergency procedures), NAVIGATOR (navigation sensor management, navigation error state estimation, dynamic route planning), COMMUNICATOR (radio/data operations), OBSERVER (weather, air traffic, inbound armament lookout and alarm, Information Friend or Foe (IFF)) ATTACKER (offensive weapon control, stores management, target acquisition/prioritization), DEFENDER (defensive weapon control, stores management), and SPOOFER (Electronic Measures (EM) and Electronic Counter Measures (ECM)). The ninth rule-based system (EXECUTIVE) coordinates mission-specific tasks and has knowledge of the mission plan. For the configuration shown in Fig.1, the human Pilot of the aircraft acts in the capacity of mission coordinator. She monitors the AUTOCREW systems and attends to mission planning and problems brought to her attention by the AUTOCREW components. The Pilot communicates with AUTOCREW via a common input device such as a keyboard, voice input, or touch screen. The AUTOCREW crew members communicate with the Pilot via graphics and text displays. The Pilot has full control of the aircraft and its on-board systems. AUTOCREW can provide automatic assistance when selected.

The AUTOCREW knowledge bases were developed and implemented independently. Since this research focused on the design of cooperating rule-based systems at a macroscopic level, it is sufficient initially to incorporate a general structure into each knowledge base, outlining only major tasks. Each major task, in turn, may be further divided within a more detailed knowledge base. These smaller sub-tasks are embedded within a less-detailed knowledge-base structure, thereby enhancing organization and clarity. During the preliminary design phase of any complex operational system, ensemble modules would most likely be developed by several systems groups. A clear, effective way in which these groups communicate their knowledge-base designs is necessary. Communication of knowledge-base structure and function using graphical symbology assists in identifying areas of knowledge-base cooperation very quickly, whereas task schedules are less effective

in communicating overall system contents. The AUTOCREW components were designed using a graphical symbology, as is shown below. It also is shown that the high-level contents of each AUTOCREW component provide sufficient results for demonstrating task coordination among the nine systems.

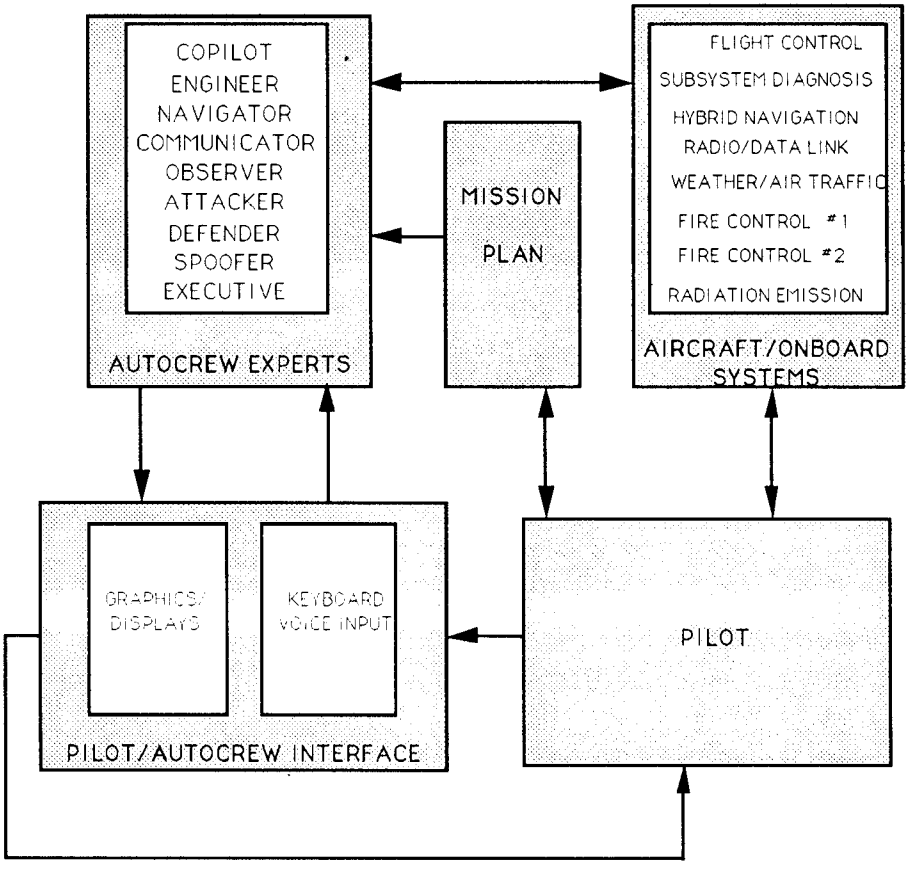

Figure 1. AUTOCREW Configuration With Pilot/Aircraft Integration.

Each AUTOCREW component is implemented as a *knowledge base*. A knowledge base is defined as a software structure that describes a system's functions and the logical relationships between the system's tasks. The relationships may be specified by *production rules* (IF - THEN statements); the production rules incorporate the AND/OR characteristics of the knowledge-base task relationships. A graphical representation of the knowledge-base structure was used to provide a first glance

overview of the system's functions. Communication of knowledge-base structure and function using graphical symbology assists in identifying areas of knowledge-base cooperation very quickly, whereas task schedules are less effective in communicating overall system contents. The basic building blocks of the knowledge-base graphical representation are shown in Fig. 2. Rectangular boxes represent knowledge-base *parameters* and ovals represent *rules*. Parameters describe factual information about the domain, whereas rules describe the procedural relationships between parameters. The "state" of a parameter is denoted by its value. Parameter values reside in slots within a parameter box in Fig. 2. Lines connecting parameters via rules form the procedural relationships between parameters. An AND relationship is denoted by a joining arc between parameters. For example, Rule R01 in Fig. 2a is read "IF the the value of parameter 1 is TRUE, AND the value of parameter 2 is value 2 THEN set the value of parameter 3 to TRUE." Each rule structure contains a premise and an action. The premise is the group of conditional clauses contained in the "if" part of the rule; the action is the group of statements that follow the "then" part of the rule. *Shared parameters* were identified throughout the design process. These "global" variables are used to exchange information between knowledge bases. A change in value of a shared parameter in one knowledge base induces search activity within another knowledge base. Hence, dynamic changes in the aircraft environment are reflected and handled appropriately within each AUTOCREW knowledge base. The graphical symbology used to represent the AUTOCREW knowledge bases can be found in [1]. The AUTOCREW system contains over 500 parameters and over 400 rules.

Knowledge Base Development of AUTOCREW Components

Although each AUTOCREW knowledge base is quite detailed, a common task organization and rule-based structure provides a logical framework. The graphical representation of this common task organization is shown in Fig. 3. As seen in Fig. 3a, there are five main task groups in each AUTOCREW crew component: 1) tasks executed during an attack on the aircraft, (2) tasks executed during aircraft subsystem emergency or potential threat situations, (3) additional tasks ordered by the EXECUTIVE, (4) tasks executed on a routine (during each search cycle) basis, and (5) mission-specific tasks which for the most part are executed once and which require a high degree of cooperation among the various systems. The assistance-to-attack and emergency-strategy tasks are executed only when search concludes that the situation warrants these actions. The routine and mission-specific tasks are executed upon each search cycle. When the search within each of the four task groups is completed (denoted by the AND relationship in Fig. 3a), the top-level parameter is set, and the cycle repeats. It is important to note that the order of execution of the four main task groups is designed intentionally. For example, if the aircraft is under attack, the situation warrants the immediate execution of aircraft and pilot-assistance measures. These tasks should be performed first before action is taken to recover from any other aircraft system emergency. The principle reason for this is that the Pilot usually has less time to react to an aircraft attack than to a subsystem alert or emergency situation. In a well-organized and highly-parallel software/hardware architecture, many emergency conditions could be addressed simultaneously. The reality of having to perform time-critical tasks in a life-threatening situation strongly motivates parallel task organization within each

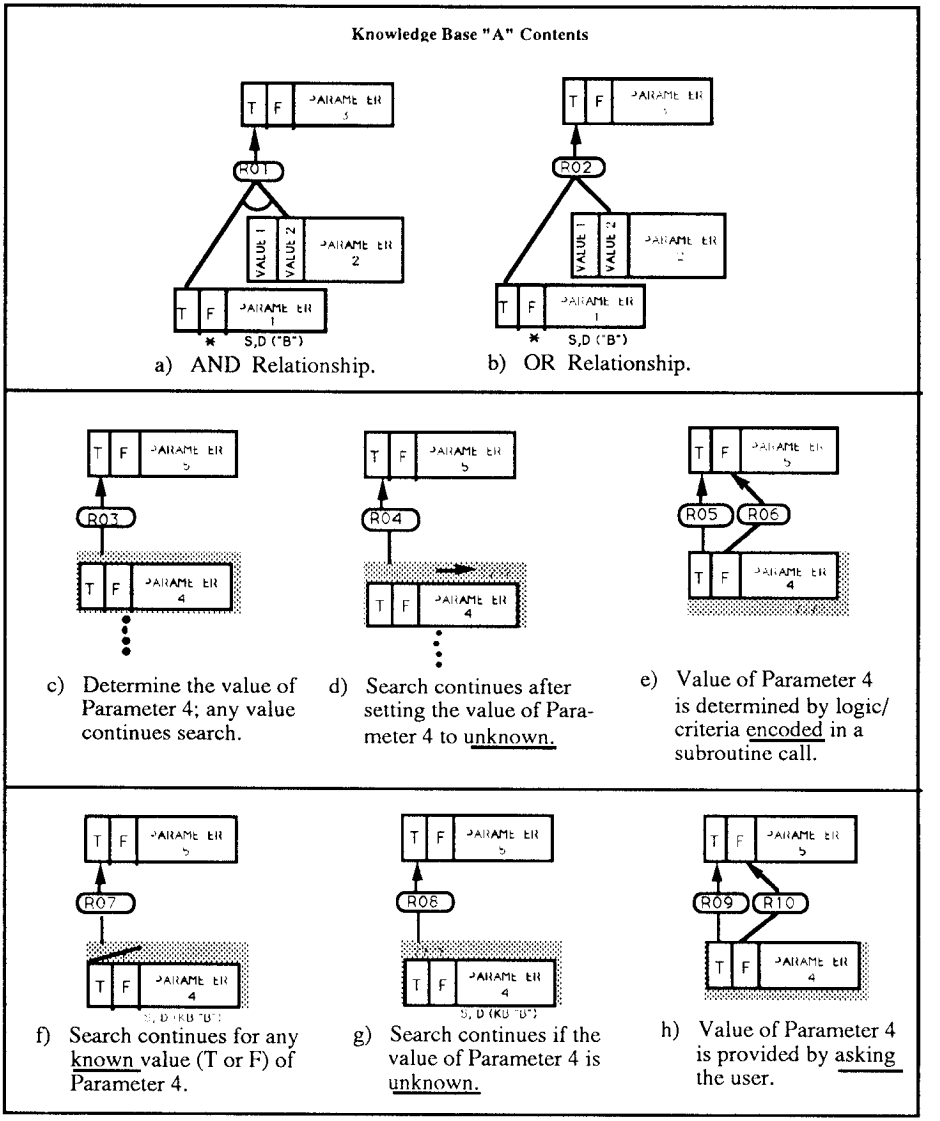

Figure 2. Basic Elements of Knowledge-Base Graphical Representation.

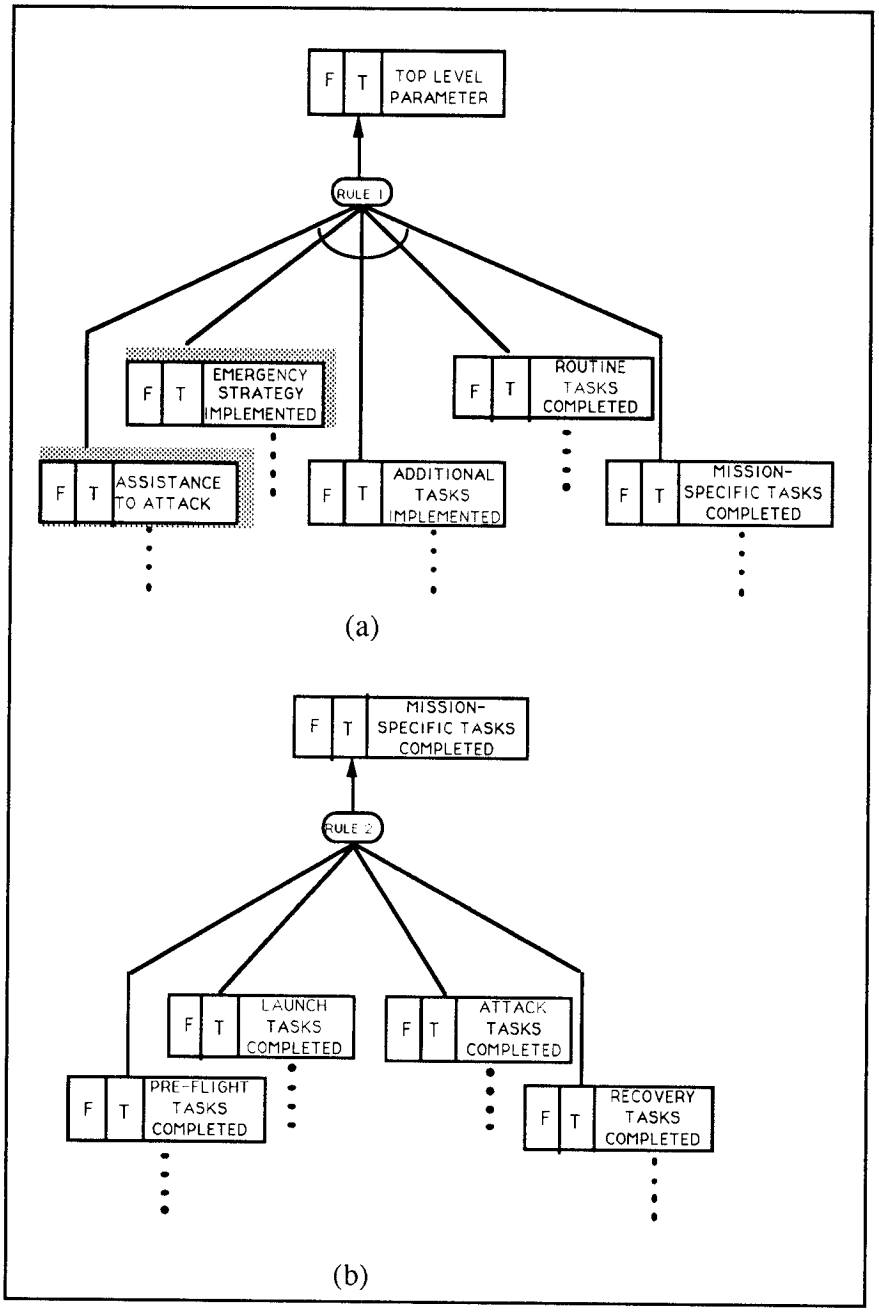

Figure 3. AUTOCREW Knowledge-Base Task Structure:
a) Complete Skeletal Knowledge-Base Task Breakdown
b) Mission-Specific Task Breakdown.

component rule-based system. If a rule-based system is designed in a serial architecture (serially executing tasks), the order in which tasks are performed may be a critical design issue, as exemplified in the air combat domain.

Mission-specific tasks are further divided into groups appropriate to each mission phase. This research focuses on the general Defensive Counter-Air (DCA) scenario. Briefly, the mission is to defend an area from aircraft attack by threatening and/or engaging incoming enemy attack aircraft. The four phases of the DCA mission are shown in Fig. 3b; The parameter, MISSION-SPECIFIC TASKS COMPLETED, in Fig. 3b is set to TRUE when any set of the mission phase tasks are completed (the OR relationship in Fig. 3b). Once again there is a deliberate order to the mission phase tasks. The pre-flight phase is dedicated to system checks and procedure reviews. The launch phase is concerned with those functions performed during and immediately after take-off. The attack phase of the mission deals with flying to the target area, acquiring and prioritizing the targets, preparing the offensive fire control system, and engaging the targets. Finally, the recovery phase pertains to the aircraft's return to base, approach, and landing tasks.

The information used in the development of the AUTOCREW knowledge bases was obtained from [16-18] in addition to many articles related to air combat in various trade publications [1]. Although the knowledge bases do not contain the detail that would be present in a fully-developed operational system, the skeletal contents exemplify the general types of tasks one would expect in a real system. As noted above, the purpose of this research is to illustrate an approach to designing a multiple cooperating knowledge-based system. Details of the AUTOCREW COPILOT system will first be discussed, followed by a discussion on implementation details, prototyping tools, and test scenarios.

Description of AUTOCREW COPILOT

The COPILOT is responsible for flight control tasks throughout the mission. The NAVIGATOR and COPILOT are the most closely tied systems; the COPILOT follows the route plan generated by the NAVIGATOR, and the NAVIGATOR takes into consideration the aircraft's performance when planning a route. The COPILOT's knowledge base is found in Figs. 4-6.

Assistance to Attack

If the OBSERVER detects an inbound enemy, the ENGINEER determines the aircraft's performance capabilities, and the COPILOT recommends (and/or selects) the appropriate evasive maneuver, as shown in Rule C04 in Fig. 4 [19]. From Rule C05 it is seen that when the ATTACKER acquires an enemy target, the NAVIGATOR generates a route to that target, and the COPILOT then determines the appropriate steering commands.

If enemy aircraft are detected by the OBSERVER, the NAVIGATOR generates an alternate route to or away from the danger area depending on the mission plan and Pilot, and the COPILOT generates the appropriate steering commands and control strategy to achieve the new route (Rule C08).

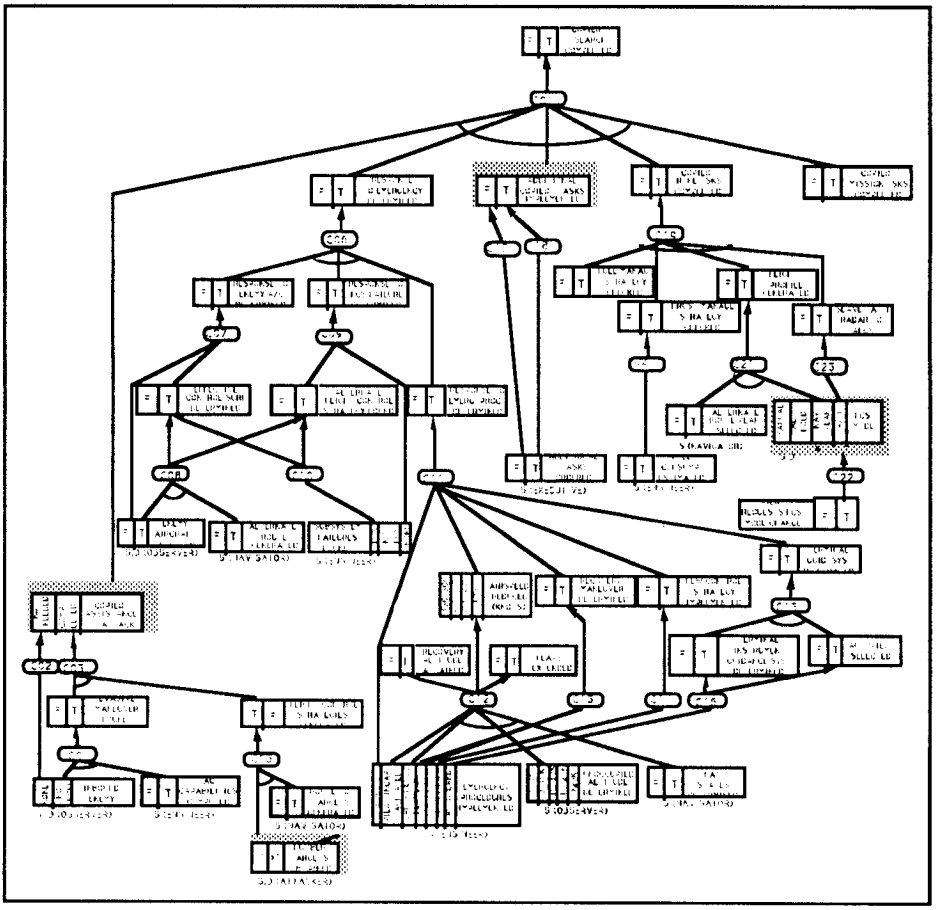

Figure 4. Skeletal Knowledge Base of AUTOCREW Member COPILOT.

Routine Tasks

The COPILOT's routine tasks consist of management, monitoring, and planning. From Rule C19 in Fig. 4, the COPILOT checks the aircraft's fuel and thrust management strategies [20]. Using the ENGINEER's fuel consumption estimates (Rule C20), the COPILOT can adjust the throttle power and control surfaces to meet the mission requirements. If the NAVIGATOR dynamically changes the flight plan or the Pilot requests a flight control mode change, then by Rule C21 the COPILOT generates another flight profile in response to the flight plan/mode change. If it is determined to operate in TA/TF mode, the COPILOT slaves the TA/TF radar to the autopilot (Rule C23).

Pre-Flight Mission Phase Tasks

The COPILOT maneuvers the aircraft into runway position upon receiving a taxi clearance through the COMMUNICATOR or Pilot as seen in Fig.5, Rule C26. The COPILOT's pre-flight tasks are completed when it has completed taxiing, and when the NAVIGATOR has generated the takeoff/departure course.

Launch Mission Phase Tasks

Not surprisingly, flight control tasks are the most numerous tasks executed during the launch phase. Hence the COPILOT becomes the most critical AUTOCREW component during this phase. From Fig. 5, the COPILOT ensures that the aircraft is in proper position on the runway upon receiving the COMMUNICATOR's tower

clearance for takeoff message. The COPILOT then computes the refusal* and rotation† speeds (Rule C29). The COPILOT waits until the ENGINEER evaluates the engine instruments and decides if the engines are operating safely. The Pilot then selects the flight control mode (manual, or according to the nav plan), and commands the takeoff when ready.

As she monitors the aircraft's speed progress, the Pilot gives the lift-off decision (Rule C30). With the NAVIGATOR's departure course established, the COPILOT maintains directional control, establishes the climb angle, reconfigures the gear and flaps, and establishes the climb speed (Rules C33 and C32). When the prespecified transition altitude is passed, the COPILOT sets the altimeter (Rule C34). Continuing its tasks, the COPILOT follows its pitch steering commands. Tactical formation and cruising speed are established, and the level-off is completed. The above description assumes that the COPILOT's knowledge base is controlling the takeoff functions. However, all, none, or part of these tasks can be executed by the COPILOT, depending on the flight control mode selected by the Pilot. It is envisioned that a COPILOT knowledge base can operate in several advisory and control modes.

Attack Mission Phase Tasks

The COPILOT's functions during the attack phase are (1) to steer the aircraft to the engagement zone (intercept), (2) to follow the Pilot's attack maneuver commands (attack), and (3) to breakaway from the attack upon weapon delivery. These events are described in Fig. 5 beginning with Rule C36. When the ATTACKER has acquired the target data and the EXECUTIVE receives final command information, the COPILOT establishes the intercept flight profile and recommends preliminary breakaway headings.

---

* The minimum speed the aircraft must be travelling at a computed location on the runway for a safe takeoff to be performed. If the aircraft speed is less than the refusal speed, the Pilot has enough runway to abort the takeoff. Some factors that are taken into account in computing this speed are aircraft type, air temperature, runway length, and runway land gradients.

† The speed at which the aircraft pitches upward on takeoff. The aircraft continues to increase its speed at the pitch angle until sufficient lift is acquired for the aircraft to become airborne.

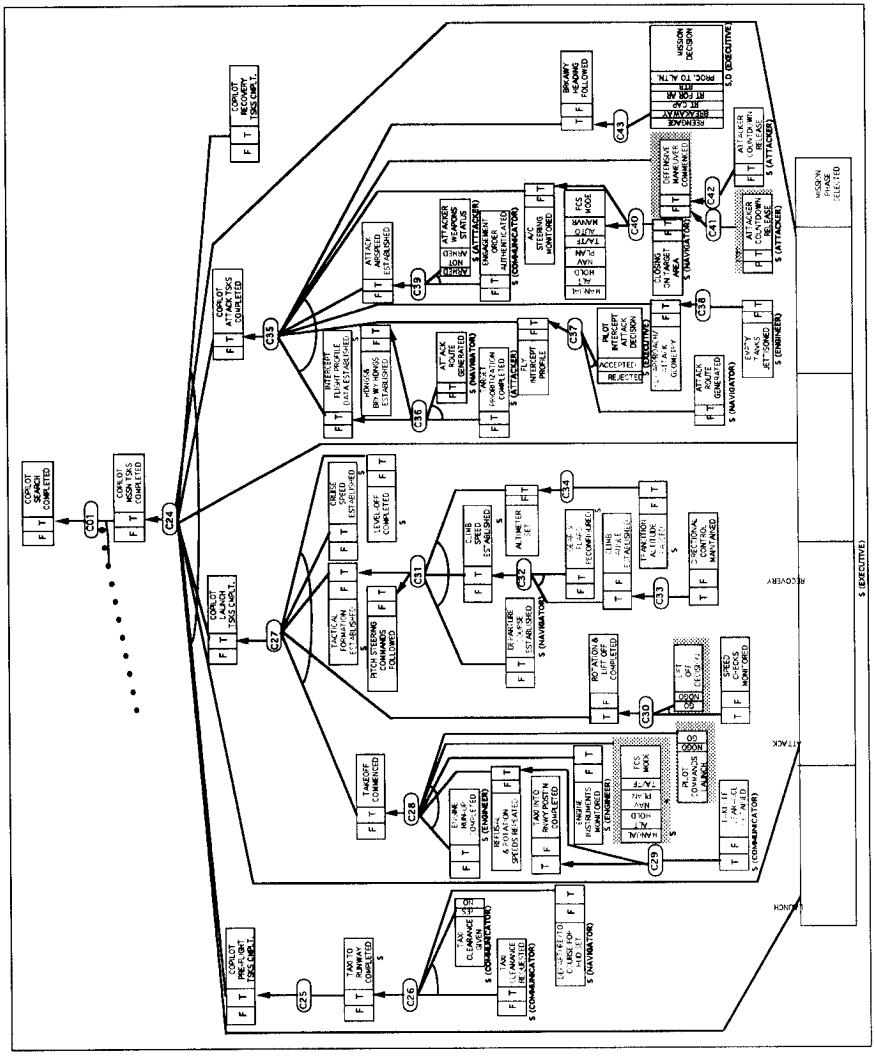

Figure 5. COPILOT Mission-Specific Tasks.

Meanwhile, the NAVIGATOR has generated a detailed attack route and the Pilot decides to accept or reject the attack. If the Pilot decides to continue the attack, the COPILOT flies the intercept profile and attack geometry (Rule C37 and C38). When the COMMUNICATOR authenticates the engagement order and the ATTACKER's weapons are armed, the COPILOT establishes the final attack airspeed, and monitors the aircraft steering commands (Rule C39 and C40). After the ATTACKER has released the weapons, the COPILOT commences its defensive maneuver away from the weapons and enemy aircraft (Rule C42). If the weapon requires infrared "target illumination" to guide it towards the target, the COPILOT implements the appropriate control maneuver. Finally, when the EXECUTIVE and Pilot decide to breakaway from the battle area, the COPILOT follows the preplanned breakaway headings.

Recovery Mission Phase Tasks

Just as the COPILOT is the most important controller during the launch phase, it plays a key role in the recovery phase. Figure 6 shows the graphical representation of the COPILOT's recovery tasks. After breaking away from the battle area, tactical formation is first re-established. If the ENGINEER determines that the aircraft is capable of safely returning to base, the COPILOT determines the optimum flight profile (altitude, speed, etc.) and selects the appropriate autopilot submodes (Rules C45-C48). The Pilot receives and acknowledges penetration instructions when she reaches "friendly" airspace. The COPILOT commences the descent and sets the radar altimeter for low height warning, by Rule C51. The penetration formation is then established, and the flaps are set for approach (Rules C49 and C50). The COPILOT then minimizes the aircraft's maneuverable speed and levels-off. On final approach, the COPILOT establishes the aircraft's final airspeed, keeping the aircraft in close formation while monitoring all flight control parameters (Rules C53 - C55). With the airbase located via the OBSERVER and NAVIGATOR, the COPILOT computes an estimated touchdown point and determines the required rate of descent (in conjunction with instrument landing system guidance). With the velocity vector thus maintained and landing checks completed by the ENGINEER, the COPILOT performs the flare out upon touchdown (Rules C56 - C58). The COPILOT then maintains directional control and spacing between aircraft on the runway, and taxis to the aircraft's final parking position (Rules C59 and C60).

Emergency Procedures

One of the most appropriate applications for an AI-based pilot aid is the implementation of emergency procedures [3,10-12]. The Pilot may not be able to react fast enough to an emergency or may fail to recall some important steps in implementing recovery measures.

The ENGINEER component of AUTOCREW is responsible for monitoring, recognizing faults, and reconfiguring the aircraft's subsystems. The ENGINEER is also responsible for declaring a subsystem emergency and implementing emergency procedures. However, the remaining AUTOCREW components can perform additional tasks to assist the aircraft in recovering from the emergency. This section describes some of the tasks that AUTOCREW would perform in the event of

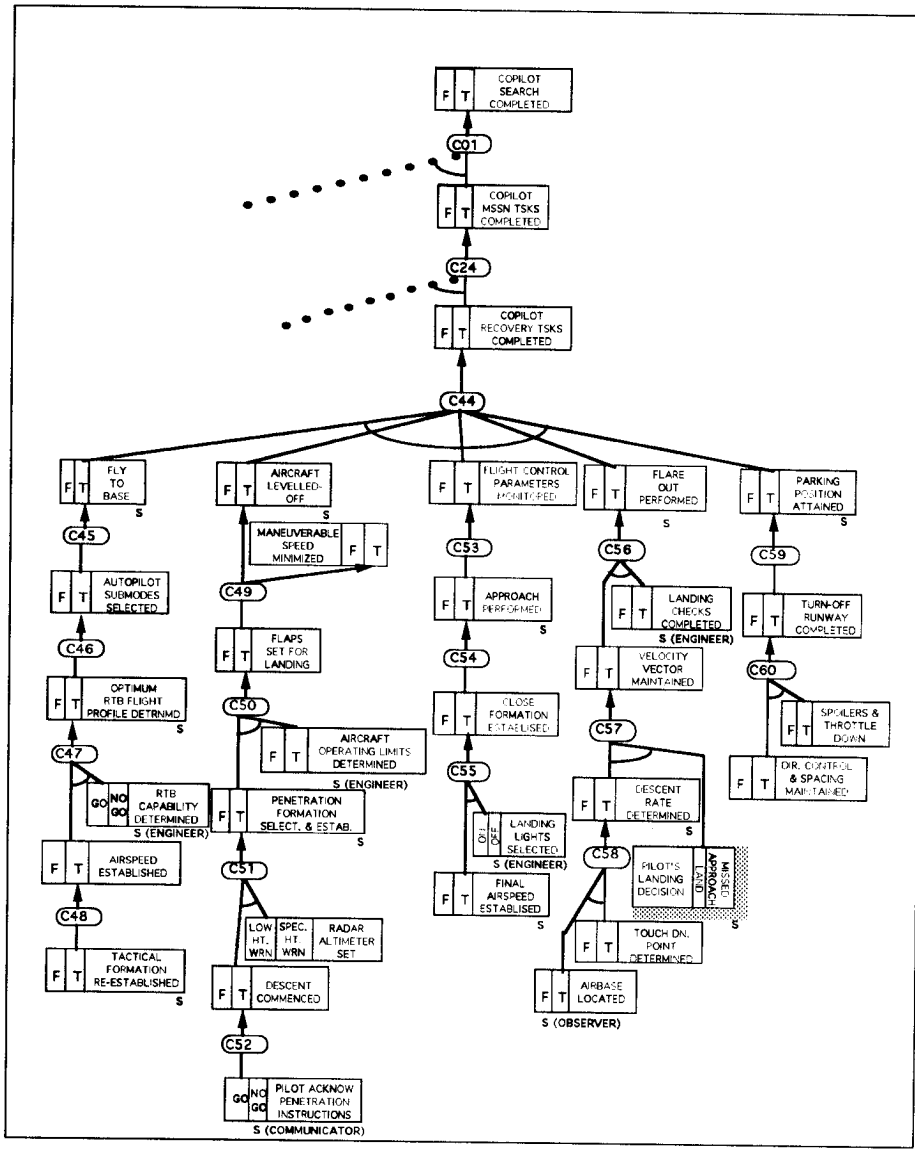

Figure 6. COPILOT Recovery Mission Phase Tasks

(1) enemy aircraft detected, (2) pilot incapacitation, and (3) engine fire. In a fully-developed operational system, the emergency procedures described in operational manuals would be automatically implemented.

Enemy Aircraft Detected

The OBSERVER's detection of enemy aircraft triggers multiple tasks to be performed by AUTOCREW in response to the emergency. If the mission phase is not attack, the DEFENDER immediately responds to the enemy detection by preparing the defensive fire control system. The ATTACKER prepares for a potential engagement with the enemy aircraft by pre-selecting an appropriate weapon and determining the target's coordinates. The SPOOFER consults with the EXECUTIVE about electronic coumtermeasures (ECM)-deployment strategy, while the COMMUNICATOR sends the detection messages to the Pilot's wingmates. After the initial detection, the OBSERVER tracks the enemy aircraft's movement and computes the time to intercept. In order for the COPILOT to choose the appropriate control strategy (Rules C07 and C08, Fig. 4) for evasive action, the ENGINEER computes the aircraft's performance capabilities while the NAVIGATOR determines the closest airfield and potential recovery route. Finally, the EXECUTIVE formulates an immediate plan and reconsiders the mission priorities in response to the unexpected enemy aircraft. Some of the information the EXECUTIVE considers in its replan recommendation are the offensive and defensive capabilities of its own aircraft, and an intelligent estimate of the enemy aircraft's tactics and motives.

Pilot Incapacitation

If a pilot is incapacitated due to a temporary blackout, AUTOCREW responds by placing all systems on automatic as dictated by the EXECUTIVE. AUTOCREW maintains control of the aircraft and "buys time" until the Pilot recovers. The ENGINEER continually monitors the Pilot's vital physiological signs. Some suggested physiological response data that potentially could be used in a loss of consciousness (LOC) expert system are (1) head position, (2) eye movement, and (3) blood pressure [21]. The LOC expert system would be embedded in the ENGINEER knowledge base. The LOC expert would sound an alarm if results showed a loss of consciousness or other form of incapacitation, and AUTOCREW would operate in an automatic mode until the Pilot recovered [12].

In order to respond to this emergency, the AUTOCREW knowledge bases execute the following tasks: The ENGINEER increases the Pilot's oxygen supply; the OBSERVER determines an unoccupied altitude [20,22]; the COPILOT descends to this altitude, reduces airspeed and extends its flaps (Rule C12 in Fig. 4); the NAVIGATOR finds the closest friendly airbase, locates friendly aircraft in the area, and generates a recovery route [23]; the COMMUNICATOR sends a message to ground control and the Pilot's wingmates informing them of the emergency; and finally, the EXECUTIVE determines the aircraft's recovery capability, decides whether an emergency landing is warranted, and orders additional AUTOCREW tasks. In the event the Pilot does not recover sufficiently to continue the mission, the EXECUTIVE could work with ground control to bring the aircraft to base autonomously (Rules C15 - C16 in Fig. 4).

Engine Fire

In the event of an engine fire, the ENGINEER closes the fuel feed pumps and shuts down the engine. The COPILOT selects an appropriate control strategy based on the expected power output (Rule C14 in Fig. 4), wind conditions, and expected aircraft

performance. AUTOCREW would then implement additional tasks to assist in recovering from the engine fire, as outlined in the aircraft's operational manual.

AUTOCREW Knowledge-Base Cooperation

As described in the previous section, AUTOCREW makes considerable use of shared parameters. Shared parameters facilitate task synchronization and cooperation between multiple rule-based systems. Table 1 shows a portion of the shared parameters found in the skeletal knowledge bases of AUTOCREW. This list does not include the shared parameters found in AUTOCREW's mission-specific task list. The table shows the knowledge base of origin for each shared parameter, and the common knowledge bases that share that parameter. The table shows that the parameters INBOUND ENEMY and ENEMY AIRCRAFT DETECTED (determined by the OBSERVER), ADDITIONAL TASKS ORDERED, and MISSION PHASE SELECTED (determined by the EXECUTIVE) are shared by each AUTOCREW component. The values of these parameters cause each system to execute a variety of tasks associated with the command or situation. In this way, a group effort is made. In considering the design and implementation details of cooperative rule-based systems, it is important to note that using a table of shared parameters is very helpful in organizing multiple rule-based systems. The table offers a first glance, summarized view of all cooperation triggers.

Table 1. AUTOCREW Shared Parameters for Emergency and Routine Tasks.

PARAMETER	KNOWLEGE-BASE PARAMETER ORIGIN									KNOWLEDGE BASES SHARING PARAMETER									
	Defender	Attacker	Spoofer	Navigator	Copilot	Observer	Engineer	Communicator	Executive	Defender	Attacker	Spoofer	Navigator	Copilot	Observer	Engineer	Communicator	Executive
DEFENSIVE CAPABILITY COMPUTED	X										X							X
DEFENDER SUCCESS	X																	X
NUMBER TARGETS OBTAINED		X											X	X			X	X
TARGET PRIORITIZATION COMPLETED		X												X				X
HOSTILE EM DETECTED			X															X
ALTERNATE ROUTE GENERATED				X										X				
ROUTE TO TARGETS GENERATED				X										X				
NAV STATES DETERMINED				X										X			X	
NAV MODE RECONFIG. RECOMMENDED				X														X
FCS MODE					X										X			
INBOUND ENEMY						X				X	X	X	X	X		X	X	X
WEATHER HAZARDS DETERMINED						X							X				X	
UNOCCUPIED ALTITUDE DETERMINED						X								X				
SUBSYSTEM FAILURES FOUND							X							X				X
EMERGENCY PROCEDURES IMPLEMENTED							X						X	X	X		X	X
AC CAPABILITIES COMPUTED							X							X				
RECEIVE COMMAND DATA								X			X			X				X
RECEIVE WEATHER UPLINK								X							X			
COM/DATA HEALTH DETERMINED								X										X
MISSION EM STRATEGY									X			X					X	
MISSION PHASE SELECTED									X	X	X	X	X	X	X	X	X	X
EXECUTIVE RESPONSE TO DEPLOY									X			X						

Summary of Development Methodology for Cooperating Systems

The methodology used in developing AUTOCREW is summarized as follows:

1) Divide each knowledge base into major task groups as shown in Fig. 3. Specifically, identify situations that need immediate attention, task groups requiring routine execution, and specialized task groups.
2) Order the task groups identified in step 1 above from most important to least important based on the time-critical nature of the task group. The inference engine chosen to infer parameter values influences the software placement in the knowledge base.
3) Break the major task groups into subtasks. The detail of subtasks built into the knowledge bases are based on the amount of detail necessary to communicate the system's functions. This is left to the designer's discretion.
4) Identify the areas of cooperation between knowledge bases. Cooperative and synchronized tasks are specified as shared parameters within the cooperating systems.
5) The functional relationships (AND/OR) between parameters in the formulation of rules should be made throughout the design process.

AUTOCREW IMPLEMENTATION

Effective demonstration, testing, and evaluation are important factors in the design of AUTOCREW. Cooperating knowledge bases can be designed on a high level to communicate the overall functionality and interactive aspects of the multiple experts. This section describes the implementation details used in the development of the AUTOCREW prototype. Several useful prototyping tools were developed to keep the multiple system design on a high level. A simulation testbed was developed to demonstrate and test the cooperating AUTOCREW ensemble's performance. Workload metrics were formulated to quantify AUTOCREW's performance in terms of the ensemble's efforts in assisting the Pilot. The combination of high-level prototyping tools and workload evaluation measures greatly reduces the time required to demonstrate and test a multiple system design, and simplifies the interpretation of simulation results.

The Princeton Rule-Based Controller (PRBC) Development System was used to implement the AUTOCREW knowledge bases. It is a unique software architecture for combining procedural and symbolic processing for rule-based system development [14]. The PRBC system is written in the IQLISP computer language [24], and many of the PRBC library functions resemble IQLISP functions. Calls to procedural Turbo PASCAL routines [25] (or any valid PASCAL code structure) can be embedded within the PRBC rule syntax. Listing 1 shows the PRBC knowledge base implementation for AUTOCREW's COPILOT, in a surface-to-air missile (SAM) attack (refer to Fig. 4 for graphical representation).

Listing 1 AUTOCREW COPILOT Listing for SAM Attack Emergency

```
[RULE_C03
  [PREMISE '($AND ($EQ EVASIVE_MANEUVER_FOUND 'TRUE)
                  ($EQ FLIGHT_CNTRL_STRAT_GENERATED
                                                'TRUE))]

  [ACTION '($SETQ COPILOT_ASSISTANCE_TO_ATTACK
                                              'COMPLETED)]
]
[RULE_C04
  [PREMISE '($AND ($NOT  ($EQ INBOUND_ENEMY 'NONE))
           ($EQ A/C_CAPABILITIES_COMPUTED 'TRUE))]

  [ACTION     '(($PASCAL "fndng_evsve_mnvrs;")
         ($SETQ EVASIVE_MANEUVER_FOUND 'TRUE))]
]
[RULE_C05
  [PREMISE  '($AND  ($EQ NUMBER_TARGETS_OBTAINED 'KNOWN)
              ($EQ ROUTE_TO_TARGETS_GENERATED 'TRUE))]

  [ACTION    '( ($PASCAL "generate_flt_cntrl_strat;")
                                        ($SETQ
FLIGHT_CNTRL_STRAT_GENERATED)) ]
]
```

The IF-THEN parts of the rules are represented by the PREMISE-ACTION pair in Listing 1. In RULE_C04, the Pascal procedure "fndng_evsve_mnvrs;" is called when enemy aircarft are detected and the Pilot's aircraft capabilities are determined. The COPILOT's assistance to an attack is completed only when an evasive maneuver is found and the flight control strategy is generated (RULE_C03). The PASCAL routine to generate a flight control strategy is called when the number of targets is obtained and the trajectories of the inbound aircarft are known. Referring to Table 1 we see that this portion of the COPILOT's knowledge base is made up of several shared parameters: INBOUND_ENEMY detected by the OBSERVER, A/C_CAPABILITIES_COMPUTED determined by the ENGINEER, NUMBER_TARGETS_OBTAINED determined by the ATTACKER, and ROUTE_TO_TARGETS_GENERATED determined by the NAVIGATOR. This is a good example of how shared parameters are used to invoke a cooperative ensemble response.

Implementing a knowledge base in the PRBC environment is achieved in two steps. The first step involves defining the parameters and rules in terms of PRBC syntax, as exemplified in Listing 1. Once the parameter-rule relations are implemented, the knowledge base's logic flow is tested in the PRBC LISP environment using PRBC's inference engine. The PRBC Development System inference engine uses a goal-directed, *depth-first search* inference engine to operate on a knowledge base. In

depth-first search, the left-most branch of an and/or graph is searched first. If the value of a goal parameter is required, the inference engine proceeds downward and to the right of the knowledge base, until enough information is inferred to determine the value of the goal parameter. The designer monitors the search logic via messages written to the screen. If the logic flow is not correct or other modifications need to be made, the designer reiterates the process until the knowledge-base search results are satisfactory. Only the logic flow is tested in PRBC's LISP environment; the embedded PASCAL calls are not executed, but instead are written to the screen as strings. The designer can, therefore, use this information to check the knowledge-base search logic. The second main step in developing a rule-based system using PRBC involves knowledge-base and inference engine translation into the final runtime PASCAL language. The PRBC system translates its special knowledge-base syntax into Turbo PASCAL code. With the inference engine also translated, the search logic determined in the LISP environment is readily available in PASCAL. The embedded calls to PASCAL routines are placed within the translated rules and are executed through control of the inference engine's search process. Since dual versions of the inference engine exist in both LISP and PASCAL, search logic can be tested in either the LISP or PASCAL development languages.

Simulating Rule-Based System Cooperation on a Single-Processor Computer

AUTOCREW simulations were run on a single-processor IBM PC-AT computer. In order to simulate multiple-system cooperation, a hierarchical framework containing all nine systems was constructed. This was achieved by combining the systems into one process containing nine independent knowledge bases. A sequential search on the top-level parameter was performed for each AUTOCREW component in a predetermined order.

In a single-processor implementation, all nine knowledge bases were initialized. Search begins with the goal of obtaining a TRUE value for the top parameter, `PROCESS ONE COMPLETED`; this occurs when the top parameters of each of the nine rule-bases is set to TRUE. On each pass through each knowledge base, search and task execution are performed for the five sub-goals as appropriate (refer to Figure 3): Assistance to Attack, Emergency Strategy Implemented, Additional Tasks Implemented, Routine Tasks, and Mission Phase Specific Tasks. Upon completing a pass through each knowledge base, all parameters are re-initialized, and the search commences again. The process is repeated until all nine systems have completed all of their tasks for the final phase of the mission (Recovery phase).

Prototype Rule-Based System Tools

The PASCAL procedures embedded within the rule structure may not exist in the designer's PASCAL library. The designer can place PASCAL procedure names in the knowledge base to describe tasks that will be developed and implemented at a later time. Unfortunately, when the knowledge base is translated into PASCAL, the compiler expects the PASCAL procedures to exist. Upon compiling the translated knowledge base, compile-time errors will occur if the referenced PASCAL procedure code does not exist. Two very useful LISP tools were developed in this research to

overcome this problem and help the designer to quickly prototype and analyze a functioning rule-based ensemble.

The first utility, called the PASCAL Routine Parser (PRP) identifies and isolates PASCAL procedures residing in the knowledge base that do not exist in a PASCAL library. Knowledge-base strings containing PASCAL code are parsed to formulate a list of PASCAL routine names. This list is then compared to procedure names found in the designer's PASCAL library. If any names in the list match the ones in the designer's procedure file, they are deleted from the list. The final list, then, contains only those PASCAL routines not found in the designer's library. The second LISP utility called the PASCAL Tag Generator (PTG), uses the list of unknown routines to generate PASCAL code shell procedures. The shell procedures are named after the unknown routine names found in the knowledge base. These PASCAL shell procedures are generated from the unknown routines found by the PRP. For the purpose of AUTOCREW, a one-line executable statement is written by the PTG in each generated procedure. This statement sends a message to the simulation graphics monitor (discussed below), displaying the name of the PASCAL routine (e.g. the name of the task to be executed) in the display area of the AUTOCREW expert that would execute the task. The set of compiler-ready PASCAL code is written to a file that is included in the final runtime PASCAL system file; hence all PASCAL procedures (including shells) embedded within the knowledge base are identified, coded, and ready for simulation. Using Listing 1 to illustrate the PRP/PTG process, the PRP would determine if the two PASCAL procedure calls "fndng_evsve_mnvrs;" and "generate_flt_cntrl_strat;" already existed in a specified PASCAL procedure library. If not, it would store these procedure names in a file. Then the PTG would write a PASCAL shell for each procedure as shown in Listing 2, and append them to the designer's library file.

The philosophy behind rapid prototyping tools is to provide a quick overview of the design without burdening the designer with excessively time-consuming details. In this application, if the designer is only concerned with the general nature of the ensemble, she should not be involved with the details until changes to the knowledge bases produce satisfactory results. In this way, the designer gets a realistic view of the ensemble component integration. The PRP and PTG prototyping tools are good examples of design aids that assist in high-level systems definition.

Listing 2 PTG-Generated PASCAL Procedure Shells

```
procedure fndng_evsve_mnvrs;
   begin
   write_to_screen('fndng_evsve_mnvrs;',
        1,new_line(copilot),copilot);
   end;{fndng_evsve_mnvrs;}
procedure generate_flt_cntrl_strat;
   begin
   write_to_screen('generate_flt_cntrl_strat;',
        1,new_line(copilot),copilot);
   end;{generate_flt_cntrl_strat;}
```

Simulation Testbed

The simulation testbed for AUTOCREW resides on an IBM PC-AT computer. In order to simulate the dynamic flight environment realistically , an interactive testbed was devised using the computer keyboard as the simulation controller. Coded keystrokes control the values of several knowledge-base parameters, thereby allowing the designer to test search activity interactively by changing parameter values. Hence, the designer can control and verify the knowledge-base logic flow and change the simulation on-line. This method is most representative of the actual dynamic flight environment. When all possibilities have been tried, the designer can make suitable adjustments to the knowledge base and continue.

The AUTOCREW display used for engineering development is shown during an inbound SAM simulation (Fig. 7). There are nine window areas, one for each AUTOCREW member. Messages are sent to the Pilot from AUTOCREW and displayed in the appropriate window area. The most useful messages during preliminary design of multiple cooperating expert systems are those that tell the designer which tasks would be performed in the fully developed system, as discussed in the previous section. When an AUTOCREW member requires attention, the window area assigned to the AUTOCREW member flashes, the task to be performed is displayed, and a unique audible tone is generated for that crew member. Although an operational AUTOCREW would surely use graphical representation of AUTOCREW activities and data, the simulation testbed gives the designer a good idea of crew member cooperation and Pilot/AUTOCREW interaction in the early development stages.

AUTOCREW WORKLOAD METRICS

The comparison of task workloads between multiple rule-based systems facilitates the design of systems having equally distributed tasks. For example, if AUTOCREW's ENGINEER has twice as many tasks to perform as the NAVIGATOR, then subdividing the former's tasks into two independent systems may be warranted. The idea of equal workloads among several rule-based systems is consistent with optimal usage of computational resources. In this research, workload metrics were quantitatively described by two *search effort metrics* called Rule and Parameter Fractions. The Rule Fraction may be viewed as a metric describing the complexity of the decision-making process in terms of subtask breakdown, while the Parameter Fraction is a reasonable measure of the quantity of tasks performed. In order to compare workload characteristics between several knowledge bases for a given mission phase, the Rule and Parameter Fractions are defined as:

$$\text{Rule Fraction}_{KB} = \frac{\text{Total Rules Fired During Mission Phase}_{KB}}{\text{Total Mission Phase Rules in All Knowledge Bases}} \quad (1)$$

$$\text{Parameter Fraction}_{KB} = \frac{\text{Total Parameters Set During Mission Phase}_{KB}}{\text{Total Mission Phase Parameters in All Knowledge Bases}} \quad (2)$$

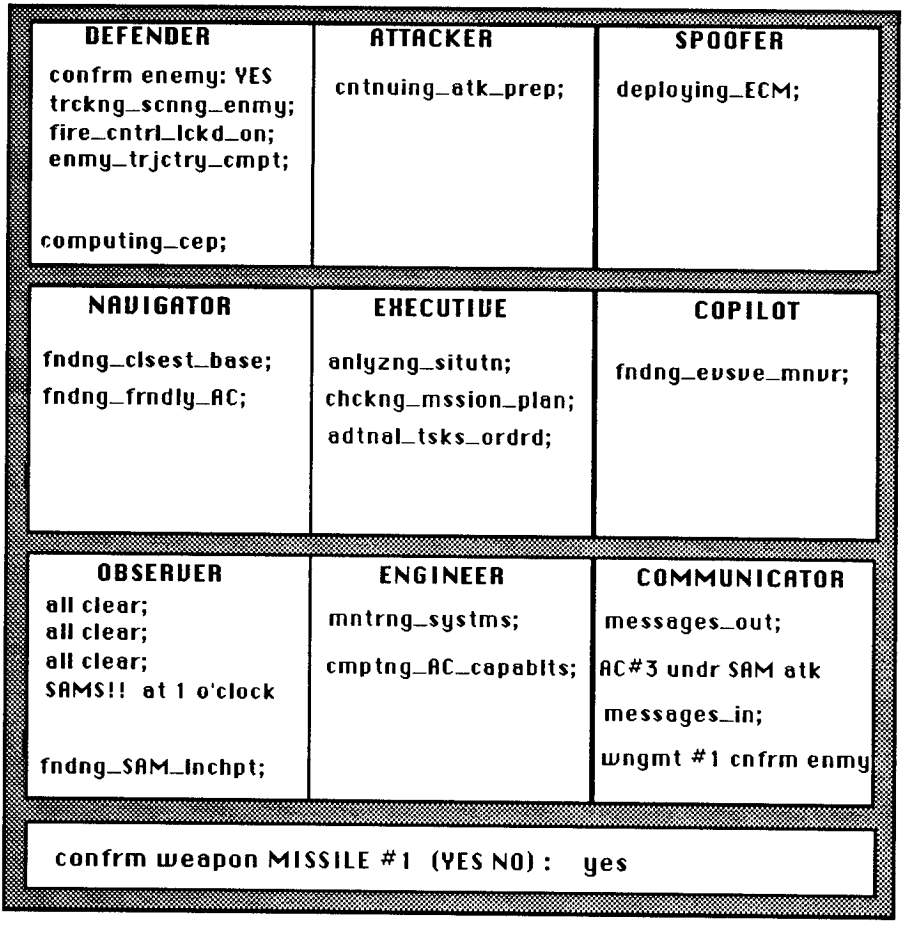

Figure 7. Engineering Development Display of Inbound SAM Simulation At Detection Time.

where the subscripts KB in the numerators denote the rules fired and parameters set in the knowledge base of interest. These workload metrics were used to show the workload distribution among the AUTOCREW components for several mission and emergency scenarios The results are shown in Figure 8. The ENGINEER, COMMUNICATOR, OBSERVER, NAVIGATOR and EXECUTIVE have the largest number of routine tasks to perform during each search cycle. During the pre-flight phase, the ENGINEER, COMMUNICATOR, and NAVIGATOR are the busiest components, as each prepares the aircraft for takeoff. About 40% of the all the launch phase tasks are performed by the COPILOT, as shown in Fig. 8b, while

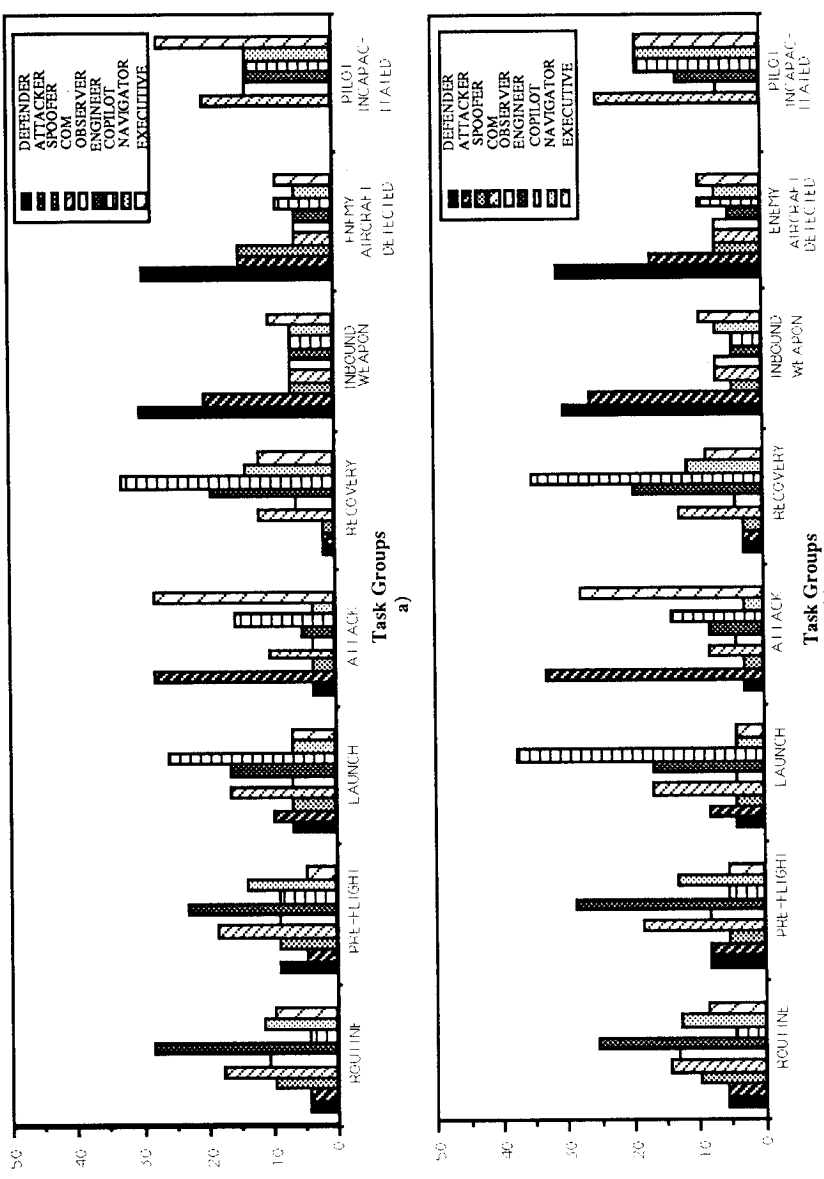

Figure 8. Workload Comparison Between AUTOCREW Crew Members: a) Rule Fraction, and b) Parameters Fraction.

the next largest task load is executed by the COMMUNICATOR and ENGINEER (each having 17% of the total launch workload). Most of the attack phase work is done by the ATTACKER (33%), EXECUTIVE (26%), and COPILOT (15%) as shown in Fig. 8b, whereas the recovery phase is dominated by the COPILOT's (35%), ENGINEER's (18%), and COMMUNICATOR's (12%) activities.

Most of the workload during an inbound weapon attack or when an enemy aircraft is detected is performed by the DEFENDER (about 30% in Fig. 8b). The ATTACKER's workload also increases at this time, as it monitors the DEFENDER's firepower capability. In the event the ATTACKER is needed to continue to defend the aircraft, it assumes the same tasks as the DEFENDER. There is evidence of more SPOOFER decision-making when an enemy aircraft is detected than when the OBSERVER detects an inbound weapon. This is due to its ECM strategy consultation with the EXECUTIVE, as noted above. The COPILOT's workload also increases during these two emergency conditions; in both scenarios, the COPILOT selects an appropriate evasive maneuver. These selections are based on the aircraft capability information provided by the ENGINEER. The EXECUTIVE's workload also increases at this time to assist the Pilot in making decisions. When the Pilot is unable to make any decisions due to incapacitation, the EXECUTIVE becomes the primary decision-maker, as shown in Fig. 8a. The COMMUNICATOR's workload increases approximately 20% above its routine tasks to relate the Pilot's circumstances to her wingmates. The NAVIGATOR's and COPILOT's specific tasks in this situation correspond to a workload increase of 16%, as shown in Fig. 8b. The ENGINEER's major task during this scenario is in the detection and evaluation of the Pilot's state of incapacitation; these tasks result in a 10% workload increase.

AUTOCREW SIMULATIONS

Simulation and comparative workload results for an inbound SAM attack on the aircraft are shown.

Referring to Fig. 7, the OBSERVER initially reports "all clear" as external sensors have not detected anything. The message is repeated upon each sensor sweep until a simulated SAM detection occurs. Then, the OBSERVER's parameter INBOUND ENEMY changes from NONE to SAMS. In the simulation, the inbound direction displayed with the "SAMS!!" message is a randomized integer value of the OBSERVER's parameter INBOUND DIRECTION. The SAMS value of INBOUND ENEMY triggers search activity within the DEFENDER knowledge base. The DEFENDER asks the Pilot to initiate the defensive fire control sequence by confirming the enemy. The DEFENDER then engages the tracking radar; the fire control system locks onto the SAM and computes the expected SAM trajectory. The COMMUNICATOR reports the SAM attack to the Pilot's wingmates and the control base. The DEFENDER calculates the Circular-Error-Probables (CEP) of the on-board defensive weapons to find the best one for SAM destruction. In addition, the DEFENDER attempts to classify the SAM using signal information obtained from the OBSERVER and its own sensors. Knowledge of the SAM classification and defensive weapon CEPs aids in the DEFENDER's selection of the best and next-best weapons. Once found, the Pilot is asked to confirm the recommended

weapon (MISSILE #1). If the Pilot inputs NO, then the next-best weapon is presented. In the event that the Pilot has armed the system and not disagreed with the selection within a given time, the DEFENDER selects the weapon and proceeds with the fire control sequence.

While the DEFENDER performs these tasks, the COPILOT defines and performs an evasive maneuver, subject to the Pilot's approval. The SPOOFER selects and deploys electronic countermeasures appropriate to a SAM, and the OBSERVER attempts to determine the SAM launch site. The COMMUNICATOR displays SAM confirmation messages from the Pilot's wingmates. The ENGINEER continues to monitor the aircraft systems and evaluates the aircraft's capabilities for the COPILOT. The NAVIGATOR locates the nearest friendly air bases and searches for friendly aircraft to provide possible fire support against the SAM site. Since the aircraft is proceeding to the target area as planned, the ATTACKER continues preparing for mission target engagement. The EXECUTIVE's responsibilities are to analyze the current mission status and to prioritize and coordinate AUTOCREW tasks. The EXECUTIVE orders additional AUTOCREW tasks to assist in mission status assessment and in achieving mission goals.

The messages in the DEFENDER's display area `"trckng_scnng_enmy"`, `"fire_cntrl_lckd_on"`, `"enmy_trjctry_cmpt"` and `"computing_cep"` are PASCAL procedures that would control and perform their implied tasks. These messages were generated by the PASCAL Tag Generator during knowledge-base development in the LISP environment. The PASCAL routines that perform these tasks remain to be developed. Hence, the cooperating rule-based system designer gets a realistic overview of AUTOCREW functions before developing the details.

This simulation illustrates AUTOCREW's cooperative task activity and team synchronization capability. Shared parameters achieve appropriate task scheduling during each mission phase. This simulation was generated from high-level definitions of the nine AUTOCREW knowledge bases implemented in the PRBC development environment and using the PRP and PTG protyping tools. The simulation shows how a potentially complex set of cooperating rule-based systems can be prototyped at a high-level, and tested for satisfactory logic flow prior to developing tasks in detail.

Comparison of Scenario Workloads

Search effort metrics also enable the designer to compare ensemble workloads for different scenarios. The increase in AUTOCREW workloads for three scenarios were compared with normal cruise task loads. The three scenarios were: 1) Cruise mode with inbound SAMs, 2) Attack mode with inbound SAMs, and 3) Cruise mode with inbound SAMs and an incapacitated Pilot. The results are shown in Table 2.

Table 2 Scenario Workload Increase Comparisons

SCENARIO	WORKLOAD INCREASE, %	
	Tasks	Decision-Making
Cruise Mode	-------	-------
Cruise Mode/Inbound SAM	26.3	28.1
Attack Mode/Inbound SAM	77.2	75.2
Cruise Mode/Inbound SAM & Incapacitated Pilot	39.5	38.6

From the table, an inbound SAM attack increases AUTOCREW's work output approximately 30% during normal cruise. If the aircraft is enroute to the air-to-air battle area, and preparing for engagement when the SAM is launched, AUTOCREW's work output increases by approximately 75% above a normal task load. This workload increase is performed in a very short period of time, so that the ratios of tasks executed over time (task rate) and information processing over time (information processing rate) are very high. These results are much higher than those for the scenario in which an inbound SAM attacks the aircraft while the Pilot is incapacitated, as shown Table 2. In comparing the first and third scenarios in the table, there is an additional increase of 10% in AUTOCREW's effort to keep the aircraft and decision-making processes under control. However, the latter results are still much lower than the results for the inbound SAM during the attack phase. The preliminary results obtained in Table 2 are reasonable because they reflect relative workload conditions that the Pilot faces during these mission scenarios. Therefore, these results can be used quantitatively to identify the critical task and decision-making areas for Pilot workload alleviation.

CONCLUSIONS AND FUTURE DIRECTIONS

This chapter outlines a general approach to designing and prototyping multiple cooperating rule-based systems. It demonstrates that high-level component design and implementation identifies ensemble requirements and facilitates system integration. The general approach used in the AUTOCREW application could be used in other areas where cooperative expert systems are needed. Examples include real-time monitoring and control of equipment in medical applications (e.g., during surgery), telecommunication network applications, manufacturing operations, nuclear power plant operations, and autonomous systems for space exploration.

Further research could be done in the area of "modes of autonomy". As illustrated in this chapter, the AUTOCREW ensemble assists the Pilot in accomplishing the mission by functioning in advisory and workload alleviating capacities. The Pilot's authority over the system is demonstrated by her continuous interaction with the ensemble and her ability to control the system's logic flow and task execution functions. The approach taken in the design of AUTOCREW achieves a variety of autonomous control modes. It is envisioned that an intelligent pilot aid would be

designed to function autonomously in the event the Pilot is unable to command the aircraft. However, it is also envisioned that in normal operations, the Pilot is the mission director and has complete authority to perform all tasks when she is capable of doing so. For example, in a normal operational mode, the Pilot would command AUTOCREW to execute lower level tasks. When the Pilot is in a high workload mode, the Pilot could command AUTOCREW to execute higher level tasks. A complete ensemble knowledge-base design is very powerful in that each knowledge base can be accessed in a multitude of ways, to provide a variety of advisory and control modes suitable to the Pilot and appropriate to the mission situation.

The AUTOCREW research showed that software tools played a key part in the successful high-level design of a complex AI-based ensemble. For example, the PASCAL Routine Parser and PASCAL Tag Generator saved much development time, and the graphically-represented AUTOCREW knowledge bases aided in the identification of shared knowledge base parameters. The simulation testbed was very useful in prototype testing. Future directions in AI-based ensemble design research should include development of additional software tools to help the designer focus on the important aspects of the system requirements.

A code-to-graph generator tool would have saved much development time. By generating graphical knowledge bases from AUTOCREW LISP code (such as shown in Fig. 4), the tool would make the design process self-documenting, saving weeks of painstaking drawing. Computer Aided Software Engineering (CASE) technology is a good candidate for graphically specifying and communicating system requirements, before software development commences. The graphical representation of knowledge bases is advocated because it provides an at-a-glance overview of the system contents. This is especially important when multiple cooperating systems are developed by several design teams and are subsequently integrated into a single complex operational system. Since the individual systems cooperate with each other, each design group must clearly communicate the knowledge-base contents so that shared information is readily identified.

The simulation testbed used to evaluate AUTOCREW's performance could be enhanced by augmenting the graphics with speech synthesis capability. Presently, messages that indicate which tasks are executed by each AUTOCREW knowledge base are written to nine different window areas, each associated with an AUTOCREW expert. During logic testing of inter-system cooperation, the messages are written to the screen very quickly. Designers would find it advantageous to have each system communicate the message in audio, with each knowledge base characterized by a different "voice".

Finally, it is important to investigate the problems associated with multiple cooperating systems using multi-processors. AUTOCREW was simulated on a single-processor IBM PC-AT computer. Simulations gave a realistic view of how the knowledge bases cooperate and share information. However, multiple systems are destined for mutli processors; Reference 26 discusses how three rule-based systems that perform cooperative tasks and share data were successfully implemented on three processors. These methods need to be applied to the

AUTOCREW system, in order to evaluate its performance in a more operationally realistic environment.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Office and U.S. Naval Air Systems Command under Contract No. DAAG29-84-K-0048, and supported by NASA and the FAA under Grant No. NGL31-001-252. The Natural Sciences and Engineering Research Council of Canada is also gratefully acknowledged for sponsoring this work.

AFFILIATIONS

Brenda Belkin is a former graduate student at Princeton University. Robert F. Stengel is Professor of Mechanical and Aerospace Engineering at Princeton University.

REFERENCES

[1] B. L. Belkin, "*Cooperative Rule-Based Systems for Aircraft Navigation and Control*," M.S.E. Thesis, Report 1856T, Department of Mechanical and Aerospace Engineering, Princeton University, June 1989.

[2] M. R. Murphy *et al*, "*A Full Mission Simulator Study of a Crew Performance: The Measurement of Crew Coordination and Decision Making Factors and their Relationships to Flight Task Performance*," 20th Annual Conference on Manual Control, Vol. 2, 1984, pp. 249-261.

[3] R. G. Eisenhardt, *Pilot's Associate Definition Study*, Perceptronics Inc., Report No. FR2213U/4577, May 1985.

[4] S. Barron and C. Feehrer, *An Analysis of the Application of AI to the Development of Intelligent Aids for Flight Crew Tasks*, NASA-CR 3944, 1985.

[5] G. Stix, "*Trends in Transportation: Along for the Ride?*," Scientific American, Vol. 265, No. 1, July 1991, pp. 94-106.

[6] J. R. Jurgensen and R. E. Feldmann, "*Expert System Pilot Aid*,", Proceedings of the IEEE 1985 National Aerospace and Electronics Conference (NAECON) May, 1985, pp. 1583-1590.

[7] E. L. Duke, V. A. Regenie and M. L. Brazee, *An Engineering Approach to the Use of Expert Systems Technology in Avionics Applications*, NASA-TM 88263, Ames Research Center, Dryden Flight Research Facility, Edwards, CA., 1986.

[8] K. Frankovich *et al*, "*Expert System Applications to the Cockpit of the '90s*," Proceedings of the IEEE 1985 National Aerospace and Electronics Conference (NAECON), 1985, pp. 1330-1335.

[9] D. A. Handelman, and R. F. Stengel, "*A Theory for Fault-Tolerant Flight Control Combining Expert System and Analytical Redundancy Concepts,*" Proceedings of the 1986 AIAA Guidance, Navigation, and Control Conference, 1986, pp. 375-384.

[10] A. D. Pisaro and H. L. Jones, "*An Expert System Approach to Adaptive Tactical Navigation,*" Proceedings of the First Conference on Artificial Intelligence Applications, IEEE Computer Society, 1982, pp. 460-464.

[11] B. M. Anderson *et al*, "*Intelligent Automation of Emergency Procedures in Advanced Fighter Aircraft,*"Proceedings of The First Conference on Artificial Intelligence Applications, IEEE Computer Society, 1982, pp. 496-501.

[12] B. L. Belkin, "*A Demonstration Expert System for Implementing Emergency Procedures in a High Performance Aircraft,*" Department of Mechanical and Aerospace Engineering Report 1749, Princeton University, Princeton, NJ, April 1986.

[13] D. C. Chen, "*An Expert Planner for the Dynamic Flight Environment,*" Proceedings of the IEEE 1985 International Aerospace and Electronics Conference, Vol. 2, 1985, pp.1347-1353.

[14] D. A. Handelman and R. F. Stengel, "*An Architecture for Real-Time Rule-Based Control,*" Proceedings of the 1987 American Control Conference, 1987, pp. 1636-1642.

[15] E. Jablonski, *Flying Fortress,* Doubleday, Garden City, New York, 1965, pp. 324-339.

[16] "*Improved Guidance and Control at the Man-Machine Interface,*" AGARD AR-228, W. M. Hollister, Editor, December 1986.

[17] R. E. Ball, *The Fundamentals of Aircraft Combat Survivability Analysis and Design*, American Institute for Aeronautics and Astronautics, New York, 1985.

[18] B. Gunston and M. Spick, *Modern Air Combat*, Crescent Books, New York, 1983.

[19] G. H. Burgin et al, "*The Adaptive Maneuvering Logic Program in Support of the Pilot's Associate Program: A Heuristic Approach to Missile Evasion,*" 24th AIAA Aerospace Sciences Meeting, Reno, NV, January 1986.

[20] A. A. Lupinetti and G. E. Long, *Future Flight, Circa 2010 A Projected Capability*, DOT/FAA Report DOT/FAA/CT-TN85/73, October 1985.

[21] R. E. Van Patten, "*Current Research on an Artificial Intelligence-Based Loss of Consciousness Monitoring System for Advanced Fighter Aircraft,*" Proceedings of the 24th Annual Symposium of the SAFE Association, San Antonio, TX, December, 1986, pp. 217-221.

[22] B. A. Bowen, "*An Expert System for Aircraft Conflict Resolution in Dense Airspaces,*" Knowledge Based Concepts and Artificial Intelligence: Applications to Guidance and Control, AGARD-LS-155, August 1987, pp. 4-1 - 4-13.

[23] M. W. Bird, "*Application of Knowledge-Based Techniques to Aircraft Trajectory Generation and Control,*" Knowledge Based Concepts and Artificial Intelligence: Applications to Guidance and Control, AGARD-LS-155, August 1987, pp. 5-1-5-12.

[24] Integral Quality, "*IQLISP Reference Manual*"," Seattle, WA., 1984.

[25] Borland International, *Turbo PASCAL V4.0 Manual*, Scotts Valley CA, 1987.

[26] D. A. Handelman and R. F. Stengel, "*An Architecture for Real-Time Rule-Based Control,*" Proceedings of the 1987 American Control Conference, 1987, pp. 1636-1642.