REPORT T-18 JUNE, 1975

CSL COORDINATED SCIENCE LABORATORY

COMPUTER-AIDED DECISION-MAKING FOR FLIGHT OPERATIONS TECHNICAL REPORT NUMBER 2

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

COMPUTER-AIDED DECISION-MAKING FOR FLIGHT OPERATIONS

Technical Report Number 2
Covering the Period January 1, 1974
to December 31, 1974

Principal Investigator: R. T. Chien

Contributors: P. Davis
K. Enstrom
P. Fitzhenry
C. Jacobus
W. Rouse
M. Selander
S. Weissman
T. Woo

This work is supported by the Avionics Laboratory,
U. S. Air Force Systems Command, Wright-Patterson
Air Force Base under Contract AF F33615-73-C-1238

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

1. SUMMARY

1.1. Introduction

The goal of this project is to construct a prototype system which demonstrates the feasibility of computer-aided decision-making in flight operations. The project consists of four phases as follows:

Phase 1. Analysis of Flight Operation and Identification of Tasks

Phase 2. Computer-Aided Decision-Making with Simple Tasks

Phase 3. Computer-Aided Decision-Making with Complex Tasks

Phase 4. Computer-Aided Decision-Making Demonstration with an Actual Aircraft Simulator

The work reported here represents progress under Phase 2 of this project. This covers the period January 1, 1974 to December 31, 1974. The technical objective of Phase 2 is to achieve the goal of constructing a simple system for the purposes of demonstrating the feasibility of computer-aided decision-making in the realm of simple tasks. This system was completed on schedule and a successful demonstration was given on February 6, 1975 at the Coordinated Science Laboratory. The purpose of this report is to document the progress achieved in some detail for later reference. During the course of this work a great deal was learned regarding problem-solving technique for computer-aided decision-making, communication problems between various programs in a large software system, and man-machine interface considerations. These insights will undoubtedly prove to be invaluable to any person or team who is interested in the development of large scale software systems in general, and to people interested in automation in particular. Based on this belief

we have included in this report not only results and achievments but also an indepth analysis of the related areas of investigation.

1.2. The Aircraft Model

In order to create a realistic environment for the development and testing of CADM software we have constructed on integrated software model functionally similar to the twin-jet aircraft and its operating enrironment. The model operates in real time and consists of three major programs -- the Subsystem Simulation, the Aerodynamic Simulation, and the Vertical Situation Display Driver.

The Aerodynamics Simulation is a digital simulation of an aircraft with four degrees of freedom. The program simulates all aircraft motions except yaw and side force. It is integrated with the aircraft Subsystem Simulation and the Vertical Situation Display program. Inputs to the Aerodynamic Simulation come from the control stick, throttles, and subsystem simulation. The aircraft simulated is a single seat, twin-jet engine, variable geometry fighter much like that proposed in the IIPACS documents. Details of the aircraft characteristics are given in Sections 2.1 and 3.1.

Bank, pitch, heading, altitude, airspeed, command altitude, and other states established by the Aerodynamic program in engineering units are input to the Vertical Situation Display (VSD) program.

The general task area for the demonstration of simple tasks for CADM was degraded mode operations. This task required a model with several constraints. The model had to simulate hardware readily identified with aircraft systems. The chosen hardware had to be capable of failure modes.

Both the pilot and CADM had to be able to control the components of the model. The resultant model was named the Subsystem Simulation.

The aircraft Subsystem Simulation model consists of two fuel tanks, four valves, two tank drain pumps, one intertank pump, two engine pumps, two thrust levers, and two engines. The valve and plumbing network allows either fuel tank to feed either or both engines. In addition, the software produces sensor outputs and failure characteristics.

Simulated failures are generated by a Gremlin program. Pumps and valves can be jammed into any current state. The fuel tanks can be empty, partially filled or full, and affect the center of gravity (C.G.) of the aircraft. The engines produce thrust proportional to fuel flow subject to the following Gremlin induced failures: engine destruction with no thrust regardless of fuel flow; engine fire with reduced thrust available; and flame out with no thrust.

The combined effect of the Subsystem Simulation, the Aerodynamic Simulation and the VSD simulation is only a first order approximation to what really is found in a modern twin-jet aircraft system, but even this crude approximation can be a challenge for pilot and CADM software. If the Gremlin and a side task are in full effect the pilot is more than occupied and is in need of assistance, the environment demonstrated the crucial need for thought in the design of man-machine interface (cooperative intelligence) software and an efficient data management system.

1.3. Computer-Aid Decision-Making For Flight Operations

The purpose of this project is to develop the conceptual framework

of a computer-aided system to relieve the pilot from high workload in flight operations. This objective is achieved by designing a system with programmed intelligence to assist the pilot in various tasks such as fuel management, weapon delivery, communications, navigation, and degraded mode operations. The emphasis during Phase 2 is in degraded mode operations. This emphasis was chosen because of a number of reasons. First, such a system provides the pilot with an increased level of confidence. Such a system also incorporates many of the necessary ingredients for other areas of study, namely, the processing of sensory data from various components in an aircraft, the identification of tasks, and the execution of tasks. Because of the inherent combinatorial complexity it is not possible to approach it in an exhaustive way. In order to achieve the desired capability, artificial intelligence techniques are applied. Among the techniques used are pattern-directed invocation of procedures, demons and self-generated special purpose procedures. Using these techniques a system is successfully implemented with a flexible control structure which is capable of performing the functions of 1) the detection of failures, 2) the correction of failures and 3) the monitoring of the effects of the correction procedure to determine its degree of success.

The detailed characteristics of a failure depends heavily on the context. The correction procedure employed takes into account contextual information, pilot presence and degree of degradation.

The control structure is designed to possess the following factors:

1) That "most critical" failure is attended first;

2) correction should be attempted in such a way as to minimize conflicts of purposes as well as needed equipment;

3) that CADM be capable of recognizing pilot actions and intentions, and to act in concert with them.

CADM maintains a list of all failures that is presently being corrected. It also monitors progress closely.

The system is flexible enough to operate in a dynamic, real-time environment. It is capable of correcting multiple failures. It tries to minimize CADM internal conflects and conflicts with the pilot. Internal conflicts are resolved using a flexible priority structure. The program is able to decide what failures have occurred, order the failures according to a easily modifiable priority scheme, and select a correction measure with respect to the available resources.

This system incorporates a flexible, multi-level control structure capable of handling a large class of problems requiring decision making. New error correction and detection procedures for new failure types can easily be added to the system by a programmer or CADM itself.

1.4. Other Developments

Although the successful demonstration in Phase 2 is primarily a demonstration of the system's capability of decision-making in degraded mode operations, a number of other advancement have been achieved. These achievements are instrumental to our success and they are quite useful in any large system where the communication between a large number of computer programs become essential.

The success of the system, in real time, depends upon the timely transfer of data and information between the subsystem and the CADM program. The exhibition of the solution depends on the timely transfer of the status to the display program. These are but two isolated examples of the very high degree of interaction necessary for the coordinated intelligent behavior of the system. The successful implementation of our system and its speedy decision-making performance is a direct result of our ingenious organization of the overall system and the solution of the communication problem between programs. We believe these technique are of direct usefulness to other systems of similar magnitude.

1.5. Future Plans

During Phase 2 we have successfully demonstrated fundamental promises of an intelligent system with decision-making capabilities for degraded mode operations. The work planned for Phase 3 will be an extension both in depth and in scope.

The present failure models will be expanded to include dynamic ordering so that actions will be context sensitive. The degree of interaction between CADM and the pilot will be expanded. A more powerful control structure will be installed to improve decision-making ability in the light of uncertainty.

In addition, we will seriously investigate the addition of a suitable navigation system so that a variety of new and challenging problems can be tackled.

CADM as a problem-solver will expand its data-base and perform

logical (not probabilistic) deduction and decision making including some adaptive features for acquiring knowledge from experience.

CADM will be expected to perform with imprecise, imperfect, and possibly conflicting data. We will also enhance the symbiotic relationship between the pilot and CADM.

2. A PROTOTYPE COMPUTER-AIDED DECISION-MAKING SYSTEM FOR FLIGHT OPERATIONS

2.1. Aircraft Models

The CADM project required construction of an integrated software model functionally similar to the twin-jet aircraft and its operating environment. The model had to be capable of operation in real time and function in the laboratory for the developers of the CADM software and hardware. The model had to be simple enough that extraneous effort was not expended on its construction, yet not so simple that it was not a realistic reflection of the parameters that a CADM program needs to manipulate in actual aircraft. The ultimate restraint of staying within the capabilities of the computer operating system and transmission bandwidths available to our group also was a basis for many decisions on building the software.

Creating a real time model meant linear approximations were used whenever possible and extraneous details were omitted. Trade-offs had to be made between the programming language, the ease of using it and the efficiency of its operation. The components of our aircraft model were chosen such that the model could have a failure with a recognizable piece of hardware clearly at fault. In some cases further hardware was made available for backup or fixing a problem so that the decision maker had some alternative actions.

Three major programs -- the Subsystem Simulation, the Aerodynamic Simulation, and the Vertical Situation Display driver -- establish the accepted reality of the environment of the entire simulation against which all other models are judged: other simulations may internalize models of

what the environment is doing and will do, but only in conjunction with the basic background environment in these three major programs.

The programming environment assumed by these modeling programs is shown in Figure 1. The Gremlin is a program which takes directions from a human observer on what hardware items are to fail. Master Monitor is the interface program between the subject pilot and the simulated aircraft systems. CADM is the title of the program which performs the computer aided decision making for the pilot. The Pilot Model is the program which monitors the pilot's behavioral activities, such as joystick movement or thrust lever movement, to aid CADM in understanding what the pilot is doing. The Controls and Display are the programs which handle the VSD, thrust levers, and joystick. The shared data base is the central buss for passing information between the different programs which are required in the entire simulation. It consists of a list of integers whose order and meaning were agreed upon by the various programmers in the group before writing their software.

The simulation was built as if many more than the two actual processors were available. This was possible because of the time sharing capabilities of the PDP-10 and the universal access to the central shared data base, Aerodynamic Simulation.

2.1.1. Specifications

The Aerodynamic Simulation is a digital simulation of an aircraft with four degrees of freedom. The program simulates all aircraft motions except yaw and side force. It is integrated with the aircraft Subsystem Simulation and the Vertical Situation Display program. Inputs to the Aerodynamic Simulation come from the control stick, throttle, and the

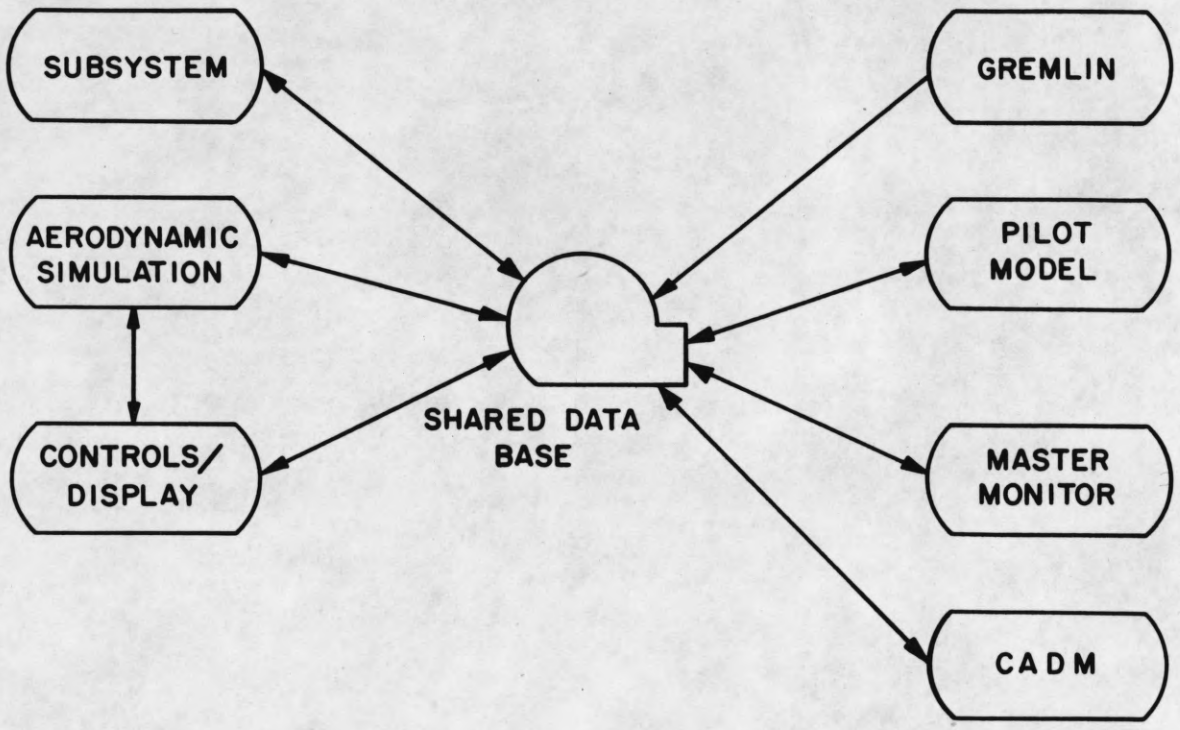

SUBSYSTEM
AERODYNAMIC SIMULATION
CONTROLS/ DISPLAY
SHARED DATA BASE
GREMLIN
PILOT MODEL
MASTER MONITOR
CADM

Figure 1

Subsystem Simulation.

The aircraft being simulated is a single seat, twin jet engine, variable geometry fighter much like that proposed in the IIPACS documents. The engines are of the 10,000 pound thrust class. Wing sweep is set at 25 degrees and aircraft speed is restricted to a range of 0.2 mach to 0.8 mach. Altitude is limited to flight levels below 10,000 feet. The aircraft is restrained from pitch angles greater than + or - 80 degrees. The bank angle is also restricted to angles of less than 80 degrees. These restraints are applied so that the linear equations of motion may be used. Since we wish only to investigate the cruise flight regime these restraints do not impede the utility of the simulation.

2.1.2. Flight Equations

Aircraft flight is governed by a series of dynamic equations derived from Newton's laws of motion. These equations are set in a reference frame that in the case of this simulation, is fixed to the aircraft. In this so-called "body axis" frame, the x axis is along the aircraft centerline running from the tail to the nose. The origin of the frame is at the center of gravity of the aircraft with the positive direction being out the nose. The y axis runs through the wings and is positive out the right wing. The z axis is positive up and is perpendicular to the xy plane. This frame is shown in Figure 2.

The center of gravity is the point where the aircraft is balanced. The aircraft is considered to maneuver around this point and it is the center of all moments. The center of gravity is the location of the aircraft's mass in point-mass analyses. When the controls (ailerons, throttles,

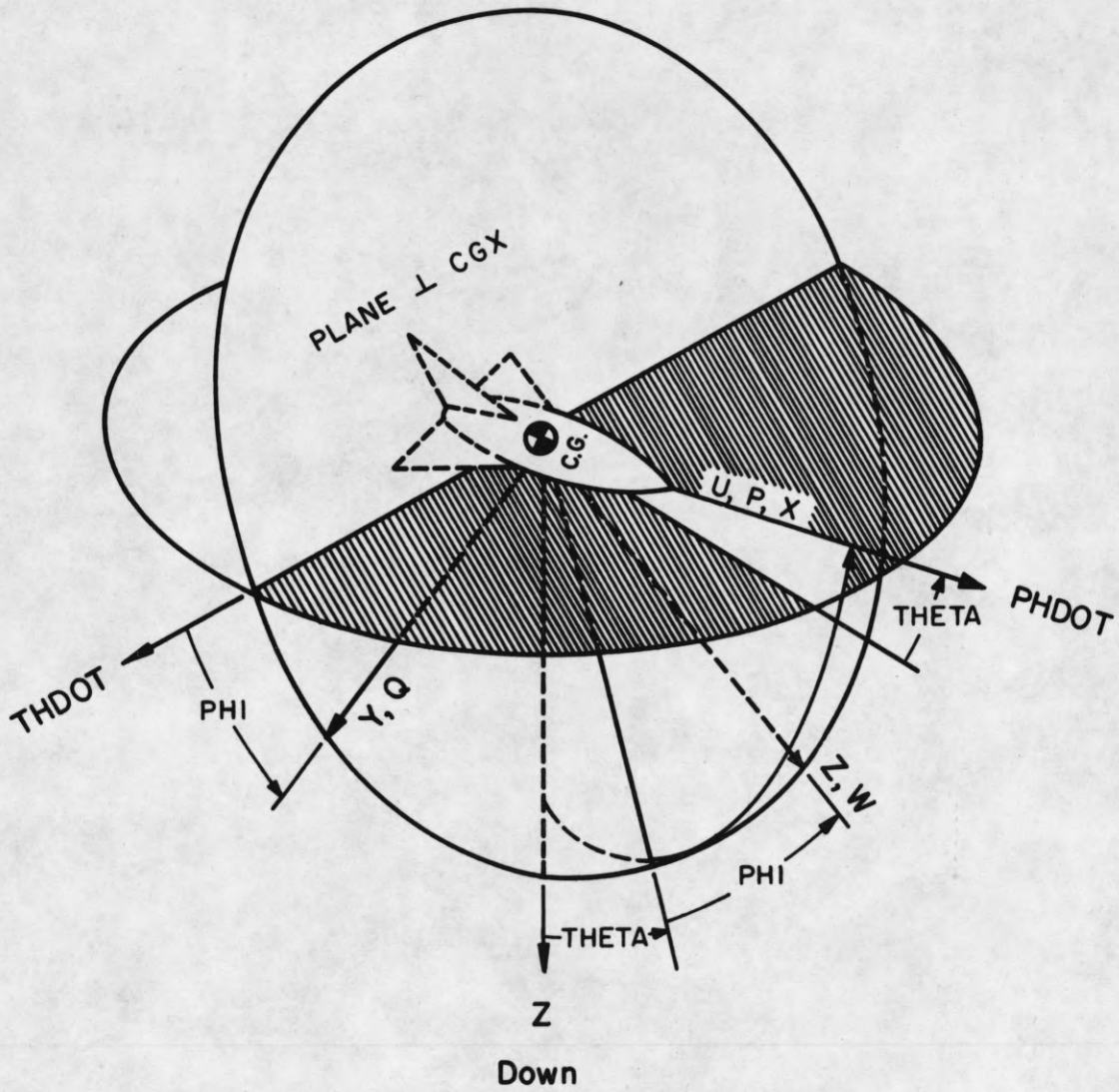


Figure 2

elevator) are activated, they create moments around the various axes. These moments cause the aircraft to maneuver around the C.G. and the aircraft will continue in this dynamic maneuver until the controls are centered. When the controls are centered, the aircraft, if it is stable, will tend to stay in the attitude it has assumed, since all the forces and moments will be balanced; thrust equals drag and lift equals weight, etc. Only the activation of the controls can change the attitude of the aircraft. Outside factors, such as turbulence can cause the aircraft to be disturbed from this equilibrium position by causing an imbalance in the forces on the aircraft and appropriate control movements must be made to rebalance the aircraft.

The sum of the forces and moments on the aircraft give the equations of motion, which when analyzed for the various components, give seven equations with 1 independent variable and 13 dependent variables. Solving these equations for the variables gives the appropriate data for display and motion in an aircraft simulator. The specific equations used are available in Section 3.1 of this report.

2.1.3. Implications and Limitations

The linear equations of flight used in the Aerodynamic Simulation allow sufficient flexibility to simulate a fairly wide range of flight conditions in the cruise flight regime. The program is an approximation of the linear equations, yet offers advantages over a more exact duplication of those equations. The program has a good representation of an aircraft in flight while neglecting many of the small or unimportant aerodynamic effects that are present in the complete equations. Thus the program is simple to code, easy to use, and can be easily modified to include any effects that

we may later wish to include.

The elimination of yaw, side forces, and side velocities restricts somewhat the utility of the simulation. The reduction in the fidelity of the simulation caused by the elimination is small in comparison to the reduction in complexity of both the code and the attendant hardware (rudder pedals, more instruments in the cockpit). For the uses we intend, i.e. cruse flight maneuvers, yaw and side forces are not considered to be important enough variables to warrant their inclusion in the simulation. If at some later point more complex maneuvers or other flight regimes are to be considered, the simulation can be modified to include the above variables, with a corresponding increase in complexity and computation time.

2.1.4. Vertical Situation Display (VSD) Simulation

Bank, pitch, heading, altitude, airspeed, command attitude, and other states established by the Aerodynamic program in engineering units are input to the Vertical Situation Display (VSD) program, which is written in BLISS-11. The VSD program recasts these data into line and alphanumeric commands for a CRT display such that the subject pilot views a display similar to a VSD. Figure 3 is a diagram of the VSD.

The program starts by building the static elements of the VSD. These are the airplane symbol, heading pointer altimeter skeleton, rate of climb skeleton. The static elements are stored in a buffer and thereafter are read only by the CRT drawing routines.

The dynamic elements are built in a separate buffering system. One buffer is built then passed on to the CRT drawing routines for display. While the CRT drawing routines are refreshing the CRT, the VSD program builds the

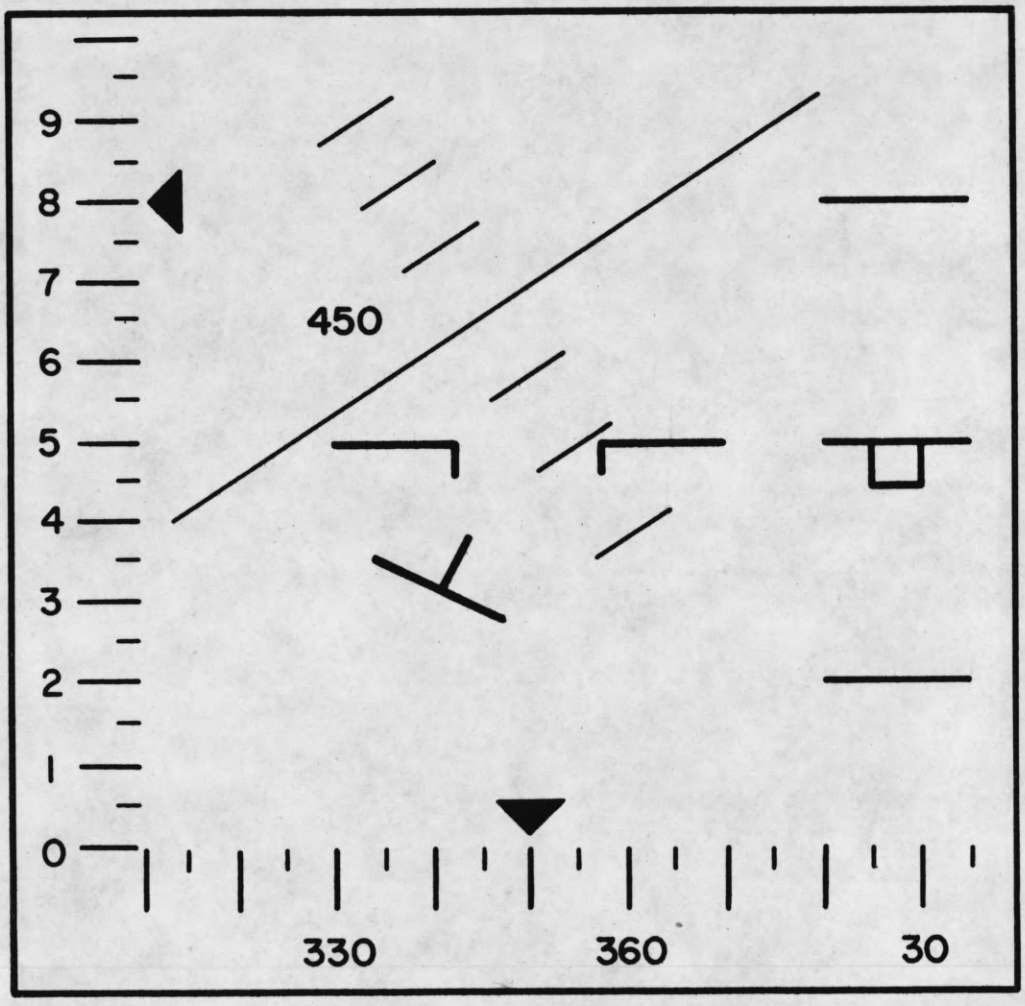


Figure 3

next dynamic display in another buffer. When this buffer is completed it is passed on to the CRT drawing routines.

The old buffer is reclaimed and the next dynamic display is built in it. This double buffering system operates continuously until the program is terminated by the operator. The dynamic section calls a data acquisition subroutine to get the required dynamic parameters for positioning the dynamic elements on the CRT. The joystick and thrust levers values are read and scaled.

The shared data base is brought in. This brings in measured thrust and center of gravity information. The Aerodynamic Simulation is called and returns the new pitch, roll, speed, rate of climb, altitude, and heading information.

Now the dynamic section starts building one of its double buffers. The compass bar is positioned under the pointer, the rate of climb bar is scaled to the appropriate length and direction, the altimeter pointer is positioned, the airspeed is printed out, the command symbol and attitude indicator are positioned. The just built buffer is then exchanged for the current display buffer and the top of the dynamic display program is again started.

2.1.5. Subsystem Simulation

The general task area for the simple demonstration of CADM was degraded mode operations. This task required a model with several constraints. The model had to simulate hardware readily identified with aircraft systems. The chosen hardware in the model had to be capable of failure modes. Both the pilot and CADM had to be able to control the components of the model.

Lastly, the model had to be simple enough that it would not exceed the available computing resources. The resultant model was named the Subsystem Simulation. For ease of programming, the computer language used was FORTRAN with some assembly language subroutines.

The aircraft Subsystem Simulation model consists of two fuel tanks, four valves, two tank drain pumps, an intertank pump, two engine pumps, two thrust levers, and two engines. The interconnection of this hypothetical hardware is shown in Figure 4. The valve and plumbing network allows either fuel tank to feed either or both engines. In addition, this software subsystem produces sensor outputs and failure characteristics.

The Gremlin program generates the failure status of aircraft at any given time and stuffs this into the shared data base. The Subsystem Simulation inputs this failure status information from the shared data base and incorporates it into the model. Pumps and valves can be jammed in any current state. The fuel tanks can be empty, partially filled, or full and affect the center of gravity (C.G.) of the aircraft. The engines produce thrust proportional to fuel flow subject to the following Gremlin induced failures: engine destruction with no thrust regardless of fuel flow; engine fire with reduced thrust available; and flame out with no thrust regardless of fuel flow.

A variety of sensors outputs are available to the other simulation programs:

1. Fuel available in two tanks.
2. Fuel flow to two engines.
3. Measured thrust from each engine.

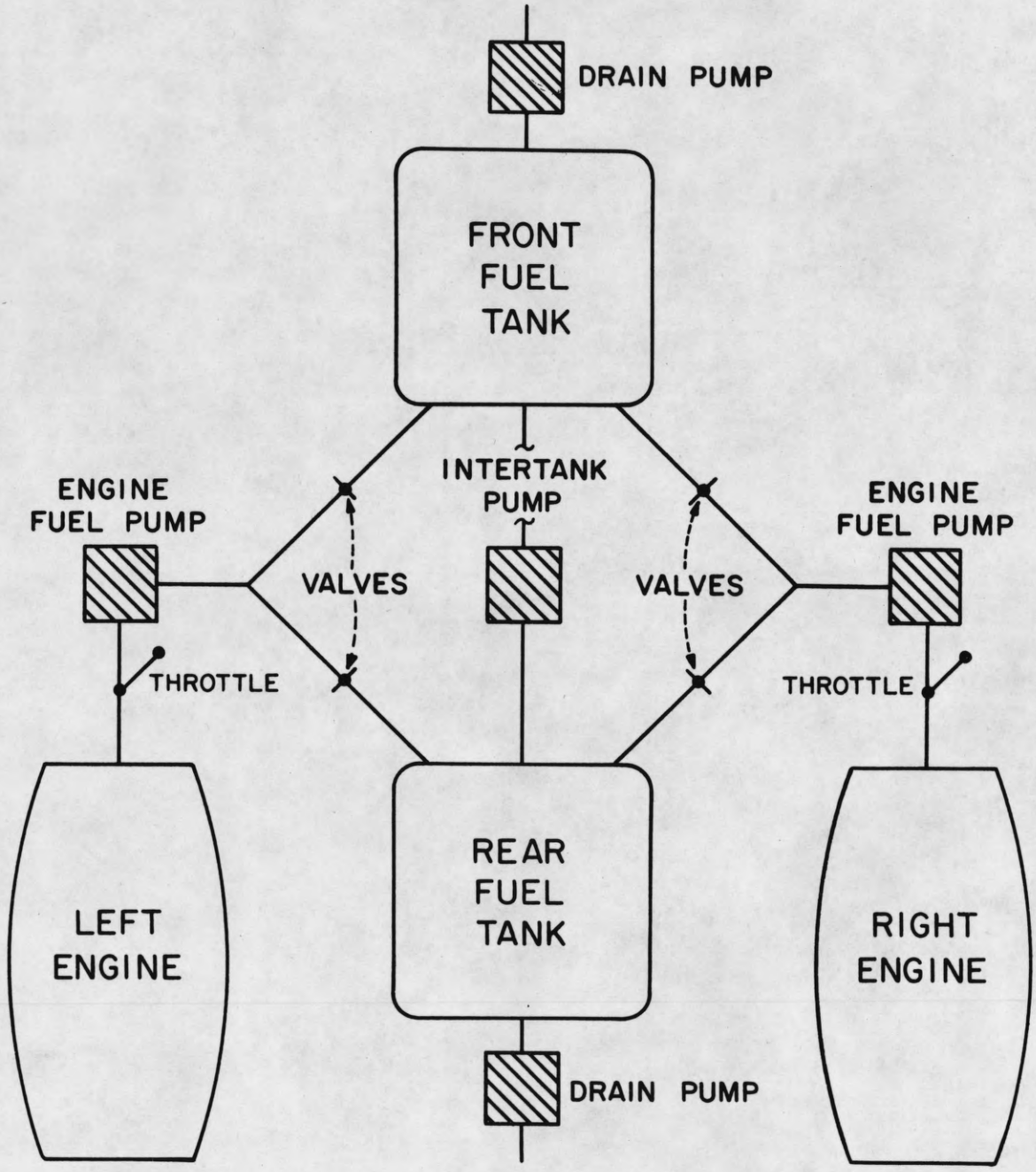

DRAIN PUMP
FRONT FUEL TANK
INTERTANK PUMP
ENGINE FUEL PUMP
ENGINE FUEL PUMP
VALVES
VALVES
THROTTLE
THROTTLE
LEFT ENGINE
REAR FUEL TANK
RIGHT ENGINE
DRAIN PUMP

Figure 4

4. Measured temperature in each engine.

5. Measure vibration modulation of each engine.

A main program controls flow of the Subsystem Simulation program by cycling through a series of calls to sub-programs responsible for emulating the various components of the subsystem. A table which summarizes the activities of the Subsystem Simulation is available in Figure 5. First the entire shared data base is read and any failures specified by Gremlin are acknowledged by setting internal flags as to the state of valves, pumps, and engines. Control settings specified by CADM or Master Monitor are implemented, if not prevented by Gremlin. For example, the CADM program may ask that a fuel drain pump be turned on. Regardless of whether the pumps previous state was on or off, the pump state will be set "on" if Gremlin has not specified that that pump is jammed. If Gremlin has jammed the pump then the pump state cannot be changed from its previous state.

The fuel flow in the plumbing network is a function of four variables. These are as follows:

1. Availability of fuel.
2. State of valves in the plumbing network.
3. State of fuel pumps.
4. Amount of demand thrust.

The subsystem fuel simulation determines what the current plumbing network will allow in fuel flow because of valve states. There are 156 possible routes for fuel flow in the network. A route for fuel is feasible if the fuel is available in the source tank or tanks, if the pump to force the fuel is on, and if the valve controlling the fuel line is open.

PHASE	FUNCTION	PARAMETER
INPUTS	STATES	PUMPS VALVES ENGINES
	COMMANDS	PILOT CADM
	CONSTRAINTS	GREMLIN
CALCULATES	FUEL SYSTEM	DIRECTION QUANTITY
	ENGINE SYSTEM	OPERATION SENSOR DATA
	INFLUENCE	GREMLIN
OUTPUTS	ENGINE SYSTEM	TEMPERATURE VIBRATION THRUST
	FUEL SYSTEM	FLOW QUANTITY LOCATION

Figure 5

For the intertank fuel line, the direction of flow is determined by the state of the intertank pump. The rate of flow is either zero or the maximum rate allowed by the intertank pump. The state of the pump is under control of the Gremlin, CADM software, the pilot via Master Monitor, or the Subsystem Simulation program itself when the auto tank leveling function is enabled.

The drain fuel lines have only one direction of flow, which is outbound from the appropriate source tank. The state of the pump is determined by Gremlin, CADM software, and the pilot. The flow rate is the maximum allowed by the drain pumps.

The engine fuel lines direction of flow is from source tanks to the engines. The state of the valves and pumps are determined by the Gremlin, the pilot, and the CADM software. The rate of fuel flow through an engine pump is zero or that rate specified by the demand thrust from the pilot's thrust lever. The fuel rate through the plumbing is zero or the rate required by the engine pumps. If two pipes have fuel flow into the same pump, each pipe carries half the fuel load.

The engine simulations are the next subprograms called with the preceding work setting up the fuel flow to each engine. The amount of actual thrust produced by each engine, called measured thrust, is equal to the demand thrust if the following prerequisites for normal engine operations are met:

1. Engine state not destroyed.
2. Engine state not fire
3. Fuel flow to the engine not zero

If the fuel flow is zero, or the engine state is destroyed, the measured thrust is zero. A destroyed engine state is nonrecoverable. A flameout, no ignition in the engine, caused by fuel starvation is recoverable. The procedure for recovery is to re-establish the fuel flow to the engine and perform an engine start function. Either the pilot or the CADM software may issue an engine start. The Subsystem Simulation responds by attempting reignition of the engine and resetting the engine start function.

The engine destroyed state is set by Gremlin. The fuel flow to the engine is established by the previously described Subsystem Software. This means that fuel starvation can occur from pilot or CADM software blunders, as well as from deliberate failure settings from the Gremlin.

The measured thrust will be half the demand thrust if the engine state is fire. The engine fire state is set by the Gremlin and is reset by turning off the fuel flow to the engine. The engine is now in a flameout state. To obtain thrust again start fuel flow back into the engine and issue an engine start function. In this simulation there is no penalty for attempting an engine start after a fire. A general clean up of state flags is performed and the specific items in the shared data base for which the Subsystem Simulation is responsible are updated. Example items are engine temperature, vibration, and fuel flow. The end of the cycle has been reached and the top of the cycle is again started.

The combined effect of the Subsystem Simulation, the Aerodynamic simulation, and the VSD simulation is only a first order approximation to what really is found in a modern twin jet aircraft system, but even this approximation can be a challenge for pilot and CADM software. If the Gremlin

and a side task are in full effect the pilot is more than occupied and in need of assistance in manipulating this first order approximation of aircraft reality. The environment demonstrates the crucial need for thought in design of man-machine interface (cooperative intelligence) software and an efficient data handling system. The modeling programs also provide a concrete vehicle for discussion of what really comprise aircraft systems and what the limitations of the equipment in these systems are.

When serious problems occur, there may be no alternatives for the pilot -- many significant failures are not correctable. Some, like having one empty fuel tank, can be "fixed," but for the most part in-air repairs are not possible. But with the aid of a CADM system, small failures can be isolated and prevented from escalating; the pilot can be alerted that some capability has been lost; and future planning for both pilots and CADM systems can take these failures into account early in the flight.

2.2. The CADM Program

2.2.1. Introduction

The Computer Aided Decision Making (CADM) program is intended to aid the pilot of the future by relieving him from routine monitoring and low level task executions. This objective is achieved by designing a computer system with "soft" intelligence -- the kind of intelligence that exhibits flexibility and accommodation for different operating environments.

The problems involved in developing a CADM to assist in various flight operations such as fuel management, weapon delivery, communications, navigation, and degraded mode operations were considered. As the project

progressed, the emphasis fell on the investigation and implementation of new techniques for degraded mode operations. This choice satisfies the project goal in many ways. First, such a system with decision making capabilities will provide the pilot with an increased level of confidence and an added capability to successfully carry out his mission. Secondly, degraded mode operations incorporate many of the desirable ingredients for other areas of study in computer aided decision making. Examples include the processing of sensory data from various components in an aircraft, the identification of tasks which CADM should carry out, and the execution of these tasks. Finally, the problem domain, because of its complexity and novelty, serves as an excellent research vehicle for developing new automation techniques for advanced aircraft.

However, because of this complexity, it would be very difficult to implement such a system using conventional techniques. The failures could occur at any time and arbitrary combinations of failures are common. It is not feasible to anticipate all of the possible combinations of failures and their proper corrections. In order to achieve the desired capability, artifi- intelligence techniques are applied. Among the techniques used are pattern directed invocation of procedures, demons and self-generated special purpose procedures. Using these techniques allows the implementation of a system with a flexible control structure in which new types of failures, detection and correction procedures can be easily added.

The problem of degraded mode operations can be broken down into several stages -- the detection of failures, correction of failures and monitoring the effects of the correction procedure to determine its success.

The major considerations in implementing these stages are the following. What are the effective detection and correction procedures for failures? If a failure cannot be corrected, how can a graceful degradation be achieved? In other words, what is the procedure for reallocating the available resources so that the failure has the least effect on an aircraft as a whole? Consequently what detection and correction procedures are appropriate in a degraded mode? These are the issues which must be resolved. Equally important is the symbiotic relationship between the CADM program and the pilot. Our program aids the pilot in a relatively "quiet" manner within its domain of capacity. An interaction occurs when high level instruction from the pilot is needed.

2.2.2. Problem Formulation

The detail characteristics of a failure depends largely on the context, i.e. the mode of operation of an aircraft. A particular combination of sensor readings may indicate a failure in an aircraft component under one mode of operation, while the same set of readings may be interpreted as "normal" under a different situation. For example, in a non-degraded situation a zero flow of fuel into an engine is considered as the consequence of a failure in either the fuel pumps or the valves. On the other hand, in a degraded mode such as one engine down or destroyed, a non-zero flow of fuel into the downed engine would represent a failure in the valves and/or the pumps. The method in which a failure is corrected is dependent upon the degree of degradation. As degradation occurs, some of the most "efficient" apparatus may become damaged or unavailable. Alternate methods of correction must be employed.

When detecting failures, it is not feasible to consider all the

possible combinations of sensor readings. For this implementation, it was not desired to employ any sort of a priori probability assignments to failures. The aim was for a system which would automatically generate and execute a set of relevant detection procedures in a flexible manner.

Along with correcting isolated failures, it is necessary to consider cases in which failures occur simultaneously or within a short time period. If a system handles failures on a first-come-first-serve basis, it is possible that correcting a failure will delay the correction of a more serious failure. CADM should recognize that the corrective procedure for a failure may be related to the procedure for others. Of particular concern is the case in which the corrective procedure for one failure conflicts with another. For example, a failure may require that a certain control be set in one state, and the corrective procedure for another failure may require the same control to be set in another state. There must be some mechanism for resolving such conflicts as well as deciding the most effective order of correction.

The pilot, as the information manager, should be informed of, yet not overloaded by, the status information. And he should have control over the results of the decision making programs at all times. In other words, the CADM programs should not be competing with the pilot for the control settings, but aid him in the background. If, for any reason, a conflict exists between what the pilot does and CADM's corrective procedures, it must be resolved in a flexible way. CADM must be aware of the pilot's actions and be able to interpret their effects on the applicability of all corrective procedures.

2.2.3. Control Structure

The CADM must be able to operate in varying environments. The environments may change because the airplane configuration can be altered. Different types of failures would mean that different corrective procedures would be necessary. To accomplish this, the control structure was implemented in a manner which allows easy introduction of new failure types and corrective procedures.

This same flexibility allows CADM to alter its own correction procedures and failure detection criteria. This means that CADM can change its action for different degrees and types of degradation. As new types of failures such as jams are determined, corrective procedures dependent on the malfunctioning apparatus could be altered or removed.

Among the performance criteria which influenced the design of the control structure were:

1) It was desired that the "most important" failures be attended to first. This necessitated the inclusion of a priority structure.
2) Without interfering with (1), an attempt should be made to correct errors in such a manner that internal conflict is minimized. When correcting two simultaneous failures, an attempt should be made to apply procedures which use different apparatus. This implies the need for a protection mechanism to indicate and reserve equipment needs for the corrections.
3) The corrections generated by CADM should be methods which involve the least conflict with any pilot actions. CADM tries to complement the pilot's actions rather than to compete with him for use of the

available equipment.

4) As a consequence of (3), CADM must be able to recognize pilot actions and determine how these actions affect failures which have been corrected and are being monitored, as well as those that are presently being corrected.

This CADM control structure is shown in Figure 6.. This program is written in MACLISP to operate on the PDP-10 computer.

When an error is detected by CADM, the program must maintain information such as who is responsible for correcting the error and how the correction is being accomplished. This information is stored on the lists as shown in Figure 1. ERROR-LIST contains all of the errors which CADM has detected but has not acted upon. These errors will be ordered from highest to lowest priority.

ACTIVE-LIST is a list of all the failures that CADM is presently correcting. Included on this list are the methods being employed to correct each failure. CADM takes the view that just because action was taken (by turning switches) which was expected to correct a failure, it cannot assume that the failure has been corrected. CADM monitors the progress of the correction procedure until a success or failure is determined.

DEMON-MESSAGE's entries indicate whether the existing error correcting procedures have succeeded or failed. The top level of CADM is able to use this information to aid in developing its future plans.

PILOT-LIST contains information concerning errors which CADM has not been able to correct. These include errors which cannot be corrected due to irreconcilable CADM-pilot conflicts or because the

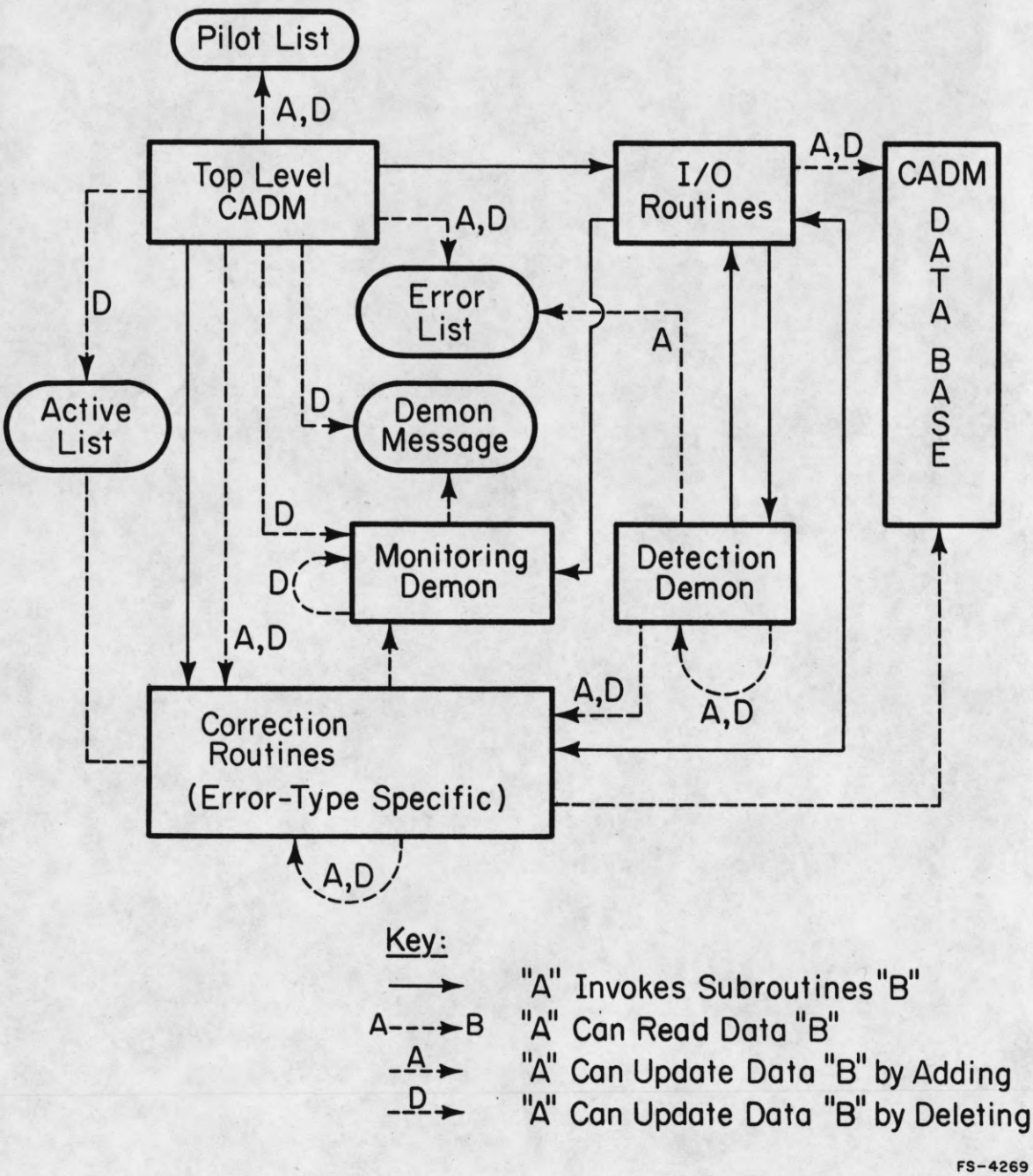

Pilot List
A,D
Top Level CADM
A,D
I/O Routines
A,D
CADM DATA BASE
D
Error List
A
Active List
D
Demon Message
D
Monitoring Demon
Detection Demon
D
A,D
A,D
A,D
Correction Routines (Error-Type Specific)
A,D
Key:
"A" Invokes Subroutines "B"
A---→B "A" Can Read Data "B"
A "A" Can Update Data "B" by Adding
D "A" Can Update Data "B" by Deleting
FS-4269

Figure 6

degree of degradation renders all of CADM's possible techniques unsuitable.

The general flow of control is independent of a specific error or error-type. When CADM inputs new data from the common CADM database, two types of information are input: sensor readings and current valve setting. CADM's detection procedures (which will be described in the next section) are used to report if any failures were detected as well as if any pilot actions have interfered with existing corrections. In the latter case, the correction procedure which was being used will be suspended and further analysis will be performed. In the former case, CADM checks to insure that the detected error is not already being dealt with by the pilot or CADM itself.

New errors cause correction routines to be invoked. A successful correction routine will alter equipment and valve settings while implementing the correction. Any conflicts with the pilot and/or CADM will be resolved at this time. When the correction routines implement a possibly successful approach, monitoring routines are established to insure that the procedure is indeed successful. Detailed discussion of the correction routines and monitoring programs will be covered in following sections. After a procedure has been implemented and is being monitored, CADM is free to correct other errors which it has already detected or to return to the common database for new data. New corrections, however, cannot use apparatus for corrections in progress without generating a conflict which must be resolved in the new error's favor.

2.2.4. Failure Detection

CADM operates in a simple aircraft domain with 20 sensor readings and 12 pump and valve settings. It is not practical to build a decision tree that incorporates all the possible combinations for all the failures. Operating

in a more complicated airplane model would necessitate a great deal of rewriting to include new sensor combinations. Much of the flexibility is lost if the detection apparatus is in the form of a big flow chart.

What is desired is a method of keying on important sensor readings and examining for failures when the readings change. It is also necessary to be able to introduce detection procedures representing new types of failures as well as altered criteria for failures, easily and in a straight-forward manner. In order to accomplish this, CADM's detection procedures are pattern-invoked. Each procedure has a pattern associated with it. A pattern is a template, a list composed of constants and variables. This means that a program can be called not only by name, but also when a datum matching the program's pattern is entered into the internal CADM database. Datum representing a sensory input can directly invoke a program. This program is referred to as a DETECTION DEMON. Because of this, sensor reading data do not have to be checked needlessly. Another advantage is that such pattern-invoked programs are easy to add into the program base without disturbing the calling sequences and branching of the logic that already exist.

The DETECTION DEMONS are of the form:

(name [pattern] [body])

where [pattern] takes the forms of the sensor reading data containing variables representing new sensor values. [body] is a decision procedure which define the criteria for detecting the error. The following is a simple example of such a pattern invoked detection program. It is used to detect flame-out on an engine.

```
(FLAME-OUT (?SIDE VIBR (RESTRICT ? FLAME-OUT))                    1
 (AND
  (PRESENT  (=?SIDE TEMP (RESTRICT ? FLAME-OUT-TEMP)))           2
  (NOT (IN (LIST  FIRE (LIST ?SIDE))  ACTIVE-LIST))               3
  (PUSH (LIST TIME  FLAME-OUT (LIST ?SIDE))
        ERROR-LIST))                                              4
```

In Line 1, a DETECTION-DEMON is defined to detect a flame-out. The body can be invoked whenever a datum list is entered into the database which has as its first element a side. In the present airplane model, ?side, a variable, would be bound to either LEFT or RIGHT. The second element of the datum must be the constant VIBR. The third element must be a numerical value which is restricted to the defined vibration range for a flame-out. Examples of data which would invoke this demon are: (LEFT VIBR 20), (RIGHT VIBR 24), (RIGHT VIBR 0). Line 2 checks for the temperature of the ?SIDE engine. The temperature must be in the flame-out range. Line 3 checks to make sure that the low temperature is not caused by a previous correction of a fire on the ?SIDE engine, which would have caused a lower temperature. This correction would appear in the ACTIVE-LIST. In line 4 the program adds to the ERROR-LIST that a flame-out has been detected.

The same type of DETECTION-DEMON structure is used to observe any pilot changes in valve or pump settings. If changes are found to have occurred, CADM can assess their relevence to failures being corrected and take appropriate action, such as suspending the correction.

2.2.5. Failure Correction

CADM examines the ERROR-LIST containing all of the recently detected errors in order to eliminate errors which are already being corrected. The entries are re-ordered so that the highest priority error

will be corrected first. CADM maintains a list of all of the error types that it recognizes along with their relative priorities. This list of priorities can be altered by the program when new information indicates that the priorities should be changed. Whenever CADM corrects a failure, the first entry on the ERROR-LIST is examined and the relevant corrective routines are invoked. At this point CADM cannot know which procedure will eventually be applied to correct the failure. It only is aware that a class of correction procedures exists. These procedures are collected into an outline, called a CPROG, for each error type. This outline is used to determine in which order the individual procedures are examined. An example of a corrective procedure is:

```
(DEFUN FLAME-OUT (X)
     (CPROG STEP1 (FLAME-OUT-RESTART)
            STEP2 (FLAME-OUT-PUMP-RESTART)
            STEP3 (FLAME-OUT-FORWARD-RESTART)
            STEP4 (FLAME-OUT-REAR-RESTART)
            STEP5 (FLAME-OUT-WRAP-UP) ) )
```

Each step in the outline represents a possible approach to correcting the problem. CADM goes through the possibilities in order searching for one which it judges to be appropriate. A correction procedure is generally comprised of three sections. First, does the state of the world represent the proper environment? In each procedure CADM wants to alter the settings for valves and pumps, but it also requires that certain valves or pumps be previously on or off. If these requirements are not met, the procedure is deemed to be unsuitable and the next procedure is examined.

Second, CADM must determine if it is free to alter the desired valves or pumps. It does this by checking to see if there are any restrictions on the required apparatus. The present CADM recongizes three types of restriction:

jam, pilot conflicts and internal CADM conflicts. In the case of a jam, altering a switch will serve no purpose, so the procedure is deemed to be inapplicable. CADM attempts to correct as many failures as it can without conflict. If a conflict is discovered, the conflicting procedure is saved, and another procedure is examined. CADM will try to resolve the conflict only if there are no non-conflict solutions.

Third, when a procedure is found which CADM wishes to apply, the valve and/or pumps are reset. In order to insure that the repair has progressed successfully, a program that monitors the correction must be constructed. This will be discussed in the following section.

If after examining all of the possible procedures, a non-conflict solution is not found, CADM returns to those in which a conflict was found and tries to resolve the conflict. The first conflicts checked are those with on-going, internally generated corrections. CADM uses the same priority structure used to order ERROR-LIST to determine which error should have control over the conflicting apparatus. If the new error has a lower priority than the existing one, the conflict will be resolved in favor of the solution in progress. If the opposite is true, the older correction (with a lower priority) will be temporarily suspended so that the new solution can be implemented. Subsequently, a new solution, not involving conflicts will be attempted.

The last type of conflict which is resolved is the pilot-CADM conflict. This is because the CADM philosophy is to lessen the pilot work load rather than to increase it. In this implementation a pilot conflict is generated when the pilot changes a relevant switch within a given time

period (in this case, within the last minute). Presently CADM does not try to interpret pilot actions, it only recognizes that these actions occur. Future implementations will try to understand the pilot's goals and reasons. (See section on Man-Machine Interaction.) In order to resolve this type of conflict, CADM asks the pilot whether it is permissible to alter a pilot controlled setting. An affirmative answer resolves the conflict in CADM's favor. A negative reply causes CADM to search for another solution. If CADM cannot correct an error it reports FAILURE if no conflicts were encountered, CONFLICT if only pilot conflicts were encountered, or nothing if only internal conflicts were present. In the last case, a correction will be attempted again when the conflict is removed.

Each of the procedures can also return a recommendation of what the next procedure should be examined. So, if a procedure determines that not only is it inapplicable, but also that the next two procedures to be investigated are also not relevant, a recommendation can be made to skip over them.

2.2.6. Monitoring Programs

When a failure is being corrected, it is not realistic to have the CADM assume that just because certain switch settings have been altered, that the error no longer exists. Rather, CADM should wait to insure that the expected response to the action occurs. However, it is not practical for CADM to stand idle during this waiting time. To avoid this, monitoring procedures are constructed for each correction of a failure at the time of correction. Each monitoring program is of the form:

```
(IF  [SUCCESS-PREDICATE]
        THEN [SUCCESS-PROGRAM]
   IF  [SUCCEEDING-PREDICATE]
              THEN [SUCCEEDING-PROGRAM]
 ELSE [FAILURE-PROGRAM]
```

SUCCESS-PREDICATE is the test for complete success. When this predicate is satisfied, the system knows that the error has successfully been corrected. SUCCESS-PROGRAM is the program to be executed when the correction succeeds. This program releases reserved equipment, activates routines which update the internal lists, and could be used to allow any desired procedure to be invoked. Consider the case in which there is a fuel imbalance. The SUCCESS-PREDICATE would specify that when the levels in both tanks are equal, the correction has succeeded. The SUCCESS-PROGRAM to be carried out after a complete success is NORMALIZE which in effect brings the control settings back to the state before the correction. If the inter-tank pump has to be turned on, it is turned off. If certain valves were turned off to force more fuel consumption from one tank, they are turned back on. SUCCEEDING-PREDICATE and SUCCEEDING-PROGRAM are similar items for the succeeding case. Normally, some bookkeeping is done while the correction in progress is succeeding. FAIL-PROGRAM is the program to execute when the correction does not succeed.

These detection procedures are to be invoked after a certain amount of time has elapsed. This can be accomplished by allowing the monitoring procedure to be invoked by new values of time. The pattern invoked mechanism which was employed during detection of failures is also used here. A demon known as a MONITORING-DEMON is created. The pattern contains a time variable. The body contains the monitoring procedure and information to restrict the times when it can be invoked and reinvoked. When a MONITORING-DEMON has

served its purpose (either a success or failure has been observed), it is removed from the system.

After the MONITORING-DEMON has been created, the ACTIVE-LIST is updated to include the present failure, how the failure is being corrected and the name of the MONITORING-DEMON which is monitoring the correction.

2.2.7. Conclusion

A system has been implemented to aid the pilot in the detection and correction of failures in aircraft with respect to the problems found in degraded mode operations.

The program is capable of detecting and correcting a class of failures in an aircraft, such as engine flame out, engine fire, and fuel imbalance, based on the model of a twin-jet aircraft described elsewhere in this report.

The control structure must be flexible and the system must be able to operate in a dynamic, real time environment. The system has the capability to correct multiple failures. It tries to minimize CADM internal conflicts and conflicts between the pilot and CADM. Internal conflicts can be resolved using a flexible priority structure. The program is able to decide what failures have occurred, order the failure according to a easily modifiable priority scheme, select a correction measure with respect to the available resource, and generate special purpose monitoring programs for the correction.

This implementation incorporates a flexible, multi-level control structure capable of handling a large class of problems requiring decision making. New error correction and detection procedures for new failure types can easily be introduced into the system by a programmer or CADM itself.

This implementation demonstrates how the application of artificial intelligence techniques such as demons and pattern-invoked procedures allow the construction of a flexible system. Further developments in artificial intelligence, especially solution of those problems found in realistic, real-time environments would permit the implementation of CADM systems which would be very difficult to construct using conventional programming techniques.

2.3. The PILOT/CADM Interface

In this section, we consider interfacing the PILOT with CADM. There are two main issues. The first is the choice of displays and controls while the second is task allocation and the resolution of conflicts between the PILOT and CADM.

2.3.1. Displays and Controls

During this phase of the CADM project, we have concentrated on computer-aided failure detection and correction. Thus, we want to consider displays specifically for that purpose. Hughes' Master Monitor Display (MMD) [Hughes, 1974] is for display of failure detection information. Since their design was based on a thorough human factors study, we need not duplicate their work in determining the appropriate display parameters. However, several extensions of MMD are necessary for our purposes.

Since our studies of CADM are not yet being carried out in an actual aircraft, the numerous dials, gages, knobs, switches, etc. are not available for the pilot to observe sensor information and input control decisions. Thus, we have extended MMD to include a "sensors" display and a "controls" display as well as the "failure monitor" display. These

displays are illustrated in Figures 7, 8, and 9.

The sensors display gives the PILOT both quantitative and qualitative information. The actual values of thrust, fuel flow, etc. are available if needed, while the low, normal, and high information allows the PILOT to make a quick check of status without having the focus long enough to perceive the actual value.

The green, yellow, and red notation appears on all of the displays. When one of these indicators is "ON", CADM perceives the system to be performing satisfactorily without CADM assistance. Green indicates that CADM is performing some task(s), but expects no difficulty in accomplishing them. Yellow indicates that CADM is performing some task(s) and does not feel it can accomplish them, but can perform some holding action until the PILOT can divert his attention to the task. Red indicates that CADM is in trouble and needs the PILOT's assistance immediately.

The controls display shows the PILOT the current settings of his controls (except for thrust and control sticks). To change the state of a control, he presses the key with the appropriate number. At the moment, a standard keyboard is being used for such input, but we would not advocate such a device for operational implementation.

The failure monitor display shows the pilot the status of the possible failures in his aircraft. Correction as well as detection information is displayed.

All of these displays would have to be expanded and perhaps made hierarchical if an actual aircraft were being considered. In a real aircraft there are too many sensors, controls, and possible failures to put each set

Sensor Readings

	Low	Normal	High
• Engines			
Left			
Thrust		900	
Flow		100	
Temp			1900
Vibration		20	
Right			
Thrust		900	
Flow		100	
Temp		1200	
Vibration		20	
• Fuel			
Forward Tank Quantity		600	
Rear Tank Quantity		600	

Green Yellow Red

FS-4249

Figure 7

Control Settings

	Open	Close	Failed
• Pumps			
Engines			
1. Left	****		
2. Right	****		
Tanks			
3. Forward Drain		****	
4. Rear Drain		****	
5. Between	F-R		****
• Valves			
Left Engine			
6. Forward Tank		****	
7. Rear Tank	****		
Right Engine			
8. Forward Tank		****	
9. Rear Tank	****		
• Restart			
11. Left Engine	****		
22. Right Engine	****		

Green **** Yellow Red

FS-4250

Figure 8

Failure Monitor

• Engines	Detected	Correcting	Failure	Conflict
Left				
Fire				
Flameout	****			
Destroy				
Right				
Fire				
Flameout	****			
Destroy				
• Fuel Pumps				
Engine				
Left				****
Right				****
Tanks				
Forward Drain				
Rear Drain				
Between			****	
• Valves				
Left Engine				
Forward Tank			****	
Rear Tank				
Right Engine				
Forward Tank			****	
Rear Tank				

Green Yellow **** Red

FS-4251

Figure 9

on a single display. Even with the current limited size of these sets of information, a hierarchy could be used to display schematics and checklists perhaps in a manner similar to Hughes' MMD. However, we should stress CADM's decision making abilities which result in the pilot not having to resort to such low level information very often.

2.3.2. Task Allocation and Conflict Resolution

In a later discussion on the general topic of human interaction with an intelligent computer (see Section 4.2), the concept of competitive and cooperative intelligence are considered. The basic idea of cooperative intelligence is that responsibility should be allocated so that the pilot and CADM do not needlessly compete and possibly produce jointly counterproductive decisions.

The suggested approach to the design of a cooperatively intelligent system is to have CADM monitor the PILOT, as well as the aircraft, and to adapt its procedures to the pilot's actions and perceptions. Since it is not reasonable to have CADM directly ask the pilot what he is doing, we need some method of inferring the pilot's perceptions and predicting his actions. In this section, we discuss a first-cut at such a method.

Before discussing the issues involved and the approach being considered, we should pause to emphasize that we are not advocating complete elimination of direct PILOT-CADM dialogue. It may be necessary and desirable for the pilot and CADM to converse directly on major issues. However, if the pilot and CADM must directly discuss the myriad of minor issues that may arise, pilot workload may increase beyond what it was without CADM.

Continuing with the discussion of an approach to monitoring the

PILOT, we should first briefly review the PILOT's role in the demonstration system developed during this phase of the project. The PILOT has two basic tasks. The first is two-dimensional pursuit tracking of the command bar on the VSO. The second task is monitoring his subsystems for possible failures and initiating corrective actions when he deems them necessary. In the second task, he has CADM as an aid. However, this does not mean that he can ignore his subsystems since CADM may not be able to solve a particular problem or, in fact, could fail itself.

Because CADM is not infallible, the PILOT may decide to take his own corrective actions, or, quite possibly, the PILOT will instinctively react to a failure and forget CADM exists. Also, the PILOT may simply make a mistake and initiate what, out of context, would appear to be a corrective procedure. In all these situations, CADM should adapt itself to the PILOT and thus avoid competition with the PILOT.

To indirectly determine what the PILOT is doing, four sources of information are available. These include sensor readings, control settings on the MMD keyboard, joystick and thrust stick outputs, and CADM's failure perceptions. We want to integrate all of this information and infer the PILOT's perceptions and classify them into three categories.

1. The PILOT has not detected any failures,
2. The PILOT has detected a specific error (perhaps unconciously), but is not attempting to correct it, and
3. The PILOT has detected a specific error and is attempting to correct it.

Sensor readings, control settings and when the PILOT last changed

them, thrust stick outputs, and CADM's perceptions of failures are readily available. However, these all relate to the PILOT's task of interacting with CADM to monitor the aircraft's subsystems. The joystick outputs relate to the PILOT's task of controlling the aircraft. It would seem that the performance on this control task would be related to the workload placed on the PILOT by the detection and correction task.

We might use RMS tracking error as a measure of control task performance, but such a measure is very sensitive to the input commands. In other words, variations in RMS tracking errors may be due to turbulence or the PILOT changing course. Instead of using tracking errors, we have chosen to "fit" a model to the PILOT-aircraft system and use the parameters resulting from this fitting process as measures of performance.

Before discussing a specific model, we should consider some of the basic issues involved. An overriding constraint on the approach is that the model parameters must be identified in real time. Thus, the model must be simple. However, the PILOT's tasks are complicated. For example, he often takes his hand off the joystick completely to momentarily devote his attention elsewhere. Long samples might smooth over this intermittency, but would also smooth over some interesting time varying attributes of the model parameters. To use short samples, we have to heuristically discard data when the joystick is producing no output.

Besides these technical issues, we are faced with the problem that there is no agreement in the literature about what models and parameters are appropriate measures of performance. However, it is not within the scope

or mandate of this project to resolve this issue. Thus, we will work with an existing model to study the feasibility of real-time identification of its parameters and usage of those parameters as measures of performance.

The natural choice is McRuer's simple crossover model [McRuer, 1969]. This two-parameter model has been used successfully to describe the performance of well-trained subjects in one-dimensional compensatory tracking tasks. For our two-dimensional pursuit tracking task, we will assume that the model can be applied to each axis (pitch and bank) independently. Since the pitch and bank axes of the aircraft (see Section 2.1) are actually coupled, the independence assumption is not really justified. However, until we can prove the feasibility of real time identification of a two-parameter non-linear model, we will avoid any elaborations of the model.

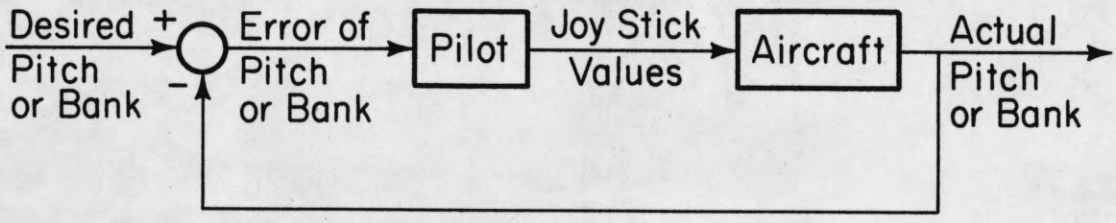
A block diagram of this approximate PILOT-aircraft system is shown in Figure 4. The desired pitch and bank angles come from the VSD, the output is the actual pitch and bank angles which are also shown on the VSD.

The discrete equations for the crossover model shown below are fit to the pilot-airplane system by measuring tracking errors and actual pitch and bank. At this point, the pilot's reaction time, is assumed to be constant and equal to .2 seconds. Thus the gains, and are the only outputs of the fitting process. Eventually the PILOT's time delay will not be assumed to be constant, and will also be an output parameter of the identification process.

$$\theta_A(t) = \theta_A(t-\Delta t) + K_\theta \, \Delta t \, \theta_E(t-\tau)$$

$$\phi_A(t) = \phi_A(t-\Delta t) + K_\phi \, \Delta t \, \phi_E(t-\tau)$$

Man-Machine System Diagram of Model Used by HUMN 10



FS-4252

Figure 10

θ_A = The Actual Airplane Bank

ϕ_A = The Actual Airplane Pitch

θ_E = Bank Error (Desired Bank-Actual Bank)

ϕ_E = Pitch Error (Desired Pitch-Actual Pitch)

K_θ = Bank Gain

K_ϕ = Pitch Gain

τ = A Time Constant Representing Pilot Delay

t = Time

Δt = A Delta Time

At this point, the modeling of the PILOT in real time requires the use of two (2) computer programs, one is located in the PDP-11 with the display programs. Here, PILOT output is rapidly sampled and stored. At prescribed intervals, the data is transferred to the second program, HUMN-10, in the PDP-10. Then, the data is analyzed to obtain gain values for the pitch and bank axes. The eventual implementation of the high speed interface should allow a single program to perform these tasks. By controlling the input data, the PILOT output gains should lie within a specified range when he is devoting his full attention to tracking the desired pitch and bank angles. If the PILOT's attention is divided among tracking and failure detection or correction, his tracking performance will degrade and should be reflected by the model gains.

By using the model gain information along with the sensor readings, control settings, and CADM's perceptions, the PILOT's perceptions are predicted.

CADM can utilize these predictions when deciding on a course of action.

A statement concerning the pilot's perceptions is only made after CADM has indicated that a specific failure has occurred. It would be extremely difficult to make a prediction without this information, since PILOT use of a specific control might be a response for a number of failure correction procedures. An example and flow diagram follows, describing the logic used in this phase to make predictions of the PILOT's perceptions.(Fig. 11)

To consider a specific example, assume that the right engine is on fire and the PILOT is initially unaware of the problem. After CADM detects the failure, HUMN-10 will check to see if the model gain outputs are abnormal or if the PILOT has applied any of the possible correction procedures. If not, the program replies that the PILOT has not detected the fire. Then, as the aircraft begins to lose power, he will probably concentrate his efforts on determining the problem source, causing his tracking performance to degrade. If this occurs within a preset time period after the failure, HUMN-10 replies that the PILOT has detected a failure, but has not begun correction procedures. After the PILOT has located the cause, he may use a number of correction procedures. He may (1) reduce his right thrust stick, (2) turn off the right fuel pump, or (3) close the valve(s) from the front and/or rear tank to the right engine. After any of these actions are performed by the PILOT, again within a preset time period, HUMN-10 states that the PILOT has attempted to correct the fire. Finally, after CADM determines the failure to be corrected, HUMN-10 resets its perceptions. If CADM is working properly, and the PILOT does not detect the problem

HUMNIO Flow Diagram

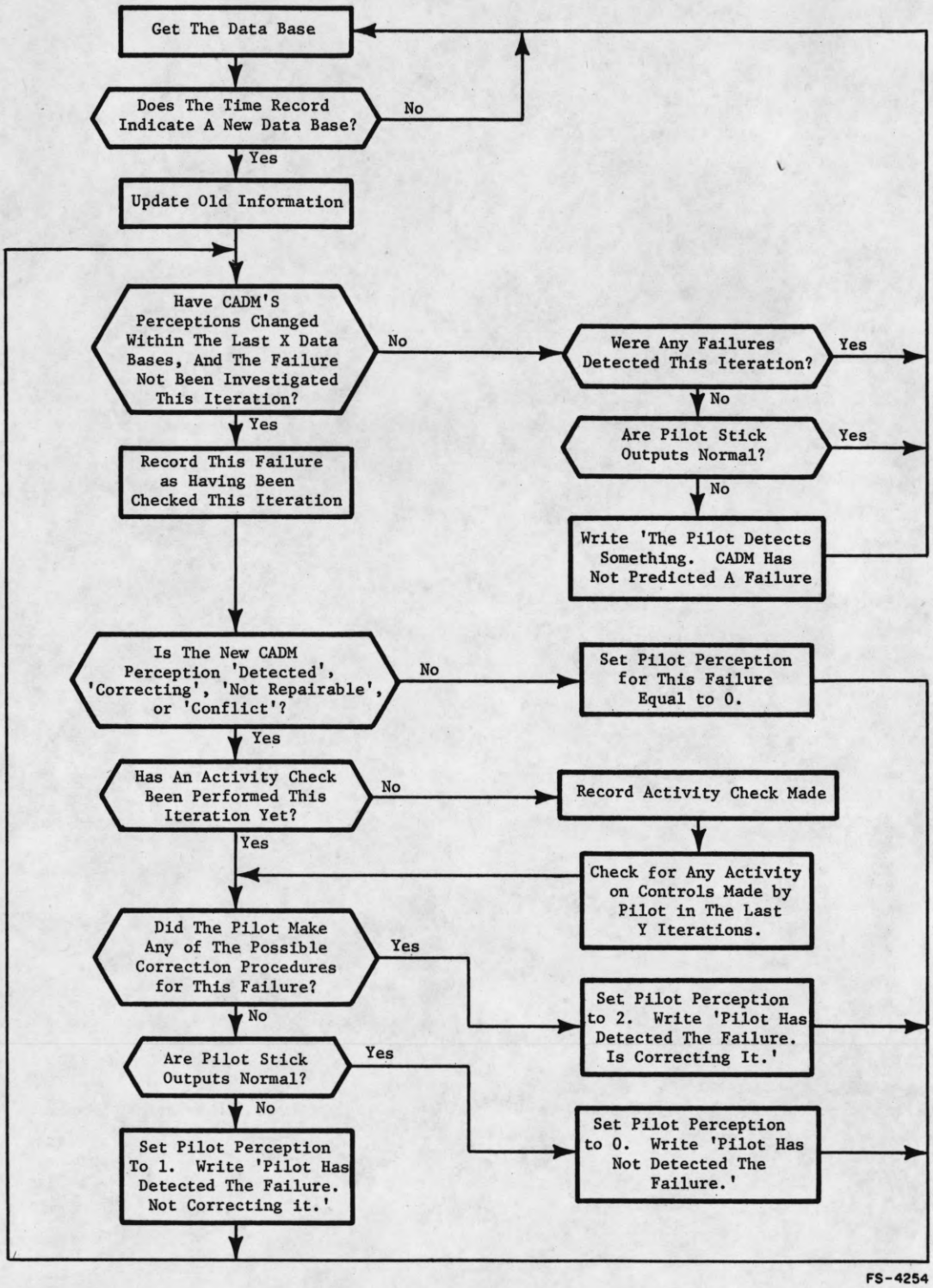


Figure 11

immediately, he may never be aware of the problem, as CADM could correct it before it is detected by the PILOT. Of course, this depends on the severity of the failure and it is unlikely that an engine fire, or at least the consequences, would go unnoticed by the pilot.

Although the monitoring method described above is rather elementary, its implementation will provide an indication of the feasibility of the general approach. Especially, we will be able to assess the constraints real-time processing imposes upon the system. Possible extensions include the use of more sophisticated PILOT models and statistical hypothesis testing.

Another possible use of the model parameters is to aid CADM in detecting failures. The PILOT may be compensating for a failure, yet not realize it exists. CADM could use this information, reflected as changes in model parameters, as part of its sensor data patterns.

REFERENCES

[1] McRuer, D. and D. H. Weir, "Theory of Manual Vehicular Controls," IEEE Transactions on Man-Machine Systems, Vol. MMS-10, No. 4, pp. 257-291, 1969.

[2] Hughes Aircraft Co., Master Monitor Display Study, Hughes Aircraft Report No. P73-464, January 1974.

3. FURTHER TECHNICAL DISCUSSION

3.1. Communications

One major impediment in the development of this demonstration has been the problem of establishing a simple and efficient means of inter-process and inter-processor communication. In this section, we will discuss the need for such communication, how it is currently being accomplished, and our future plans.

3.1.1. Why Communications

The need for inter-processor communication is obvious, since the graphics and part of the aircraft simulation is on a PDP-11 and the CADM and remainder of the simulation is on a PDP-10.

Not so obvious is the need for interprocess communication. The use of languages is one of the primary reasons. Some languages are better suited for certain tasks than others. For example, for problems in "reasoning" involving rather abstract symbol manipulation, LISP and is descendents are most convenient for prototype system development. BLISS is well suited for string manipulation and data management tasks. Finally, FORTRAN is well suited to "number-crunching" tasks. It is thus necessary for a number of programs, written in different languages to communicate. While this could have been accomplished by using a set of assembly language subroutines to provide the common linkage, this was not done for two reasons.

First, with a substantial number of people doing programming, it is very desirable if each person can run and debug his module largely independent of the other modules. This would be difficult to do if the

system were one program.

Second, FORTRAN and LISP contain no provision for multitasking. Since an aircraft contains several systems which operate simultaneously, it is necessary that any effective simulation of an aircraft simulate this parallelism. When not using languages with multitasking facilities built in, it is most convenient to achieve pseudo-parallelism by running each task as a separate job and letting the monitor's time-slicing provide the parallelism.

3.1.2. The Implementation

At the time of the last demonstration, communication between jobs was accomplished by a common data base located on a disk file. Since two FORTRAN jobs could not access this file without irrecoverable I/O errors, it was necessary to provide another job, MANAGER, and several more disk files, one for each job, such that access to the data base was controlled by MANAGER. The continuous opening and closing of so many disk files caused the system to run extremely slowly and in addition, beat the disk around an unacceptable amount.

The system has been modified to use a new version of MANAGER. The new MANAGER, written in BLISS, makes use of new monitor feature to pass data through core from one job to another rather than through a disk file.

MANAGER keeps a copy of the master data base in its own user space. Each job can pass messages to MANAGER requesting changes to the master data base, or requesting copies of any segment of that data base. The process is as follows:

First, the user job initializes itself by making a call to an assembly language subroutine which returns a unique identifier for the MANAGER

and initially setting up a communication link to MANAGER. When the job wants to change something in the data base, it changes the first three words in a work area to codes which MANAGER will understand and translate into action. It then copies the desired changes into the remaining words of the work area (the work area can be up to 512 words long). A call to another assembly language subroutine then creates a page in the user's address space, copies the work area into that page, and passes that page to MANAGER by swapping page maps.

Reading from MANAGER is by a similar process. The user again sets up a work area, this time only three words long. He passes those three words to MANAGER using the same subroutine call, and then waits for a response and the new data base.

This mode of access to the global data base requires about 50 ms per transactions.

Communication between the PDP10 and PDP11 is currently accomplished via a FORTRAN program which reads the 2400 baud tty line between the two computers and communicates with MANAGER using page passing. The PDP-11 side is run using the stand-alone system developed at CSL.

3.1.3. Future Work

We will incorporate the Master Monitor Display (MMD) and the PDP-10 to PDP-11 communications program (AFCOM) into MANAGER, since all three can conveniently be written in BLISS and by putting them into one program, we can cut down on the number of jobs which must communicate with manager using page passing, and consequently increase the speed.

We also plan to incorporate the PDP-11 side of the system into a

job running under the M&M operating system which is currently under development and nearing usability.

3.2. The PDP-10 to PDP-11 Communication Link

In any multi-processor system the final performance and reliability of that system is dependent on the flexibility and integrity of the inter-process communication links that can be established.

The Coordinated Science Laboratory CADM Project uses a system composed of two physical processors, a PDP-10 timeshared processor and a PDP-11 stand alone minicomputer. Several processes reside in the PDP-10. They include the high level decision maker, demon, and processes that update simulation parameters. On the PDP-10 interprocess communication is controlled by a mangement process and makes use of the new inter-job communication facilities available in the 601 montior. The PDP-11 supports a single process. This process is the aircraft simulator and is responsible for maintaining the video display, the joystick input control, and the flight dynamics.

In this section of the report we will be concerned with hardware aspect of the inter-processor communication link between the PDP-11 and the PDP-10. We will also examine some of the low level software required to control the hardware.

This communication link consists of two parts. The first part is a 2400 baud full duplex teletype link between the two machines. The hardware required for this line consists of a DL-11 asynchronous line module that is connected to the PDP-11 unibus and a port out of the DC-10

teletype controller already on the I/O-bus of the PDP-10. The PDP-10 thus "sees" the PDP-11 as an interactive user terminal, one of many already connected to the timesharing system. Similarly the PDP-11 "sees" the PDP-10 as an interactive terminal attached to its unibus. The second part of the communication link consists of a 7 megabaud communication channel, here after referred to as the channel, between the I/O-bus on the PDP-10 and the unibus on the PDP-11. This channel will be used to send large blocks of data from one machine to the other.

The 2400 baud tty line, while it is considerably slower than the 7 megabaud channel, plays a key role in maintaining the flexibility of the communication link between the two processors. The tty line allows a process on one machine to communicate with the monitor of the other machine, typically the process on the PDP-11 will give commands, or request information from the 601 monitor on the PDP-10. Thus a process on the PDP-11 can select, via the monitor, which PDP-10 process it is going to communicate with, or more importantly, if that process does not exist, the PDP-11 process may create the process on the PDP-10 by issuing a "run" command to the 601 monitor. The ability to run a job of the PDP-10 is a property of the tty lines and could only be accomplished via the 7 megabaud channel only with considerable difficulty. It could be done with an extensive patch to the 601 monitor, which would increase its size appreciably, or by having a special process on the PDP-10 to cater to the channel. This extra process would take up space in the job tables. It would have to be functionally another copy of the tty handler in the 601 monitor. To sum things up, the 2400 baud tty line has two major functions. During the operation of bootstrapping the system together, the tty line has the power to easily spawn processes on the PDP-10 side, if they do not already exist. It also has the power to destroy processes when they are no longer needed. The tty line is also instrumental in the

allocation of the 7 megabaud channel to different processes on the PDP-10.

The 7 megabaud channel was built to provide the capability of moving large blocks of data from one machine to the other in a minimal amount of time. Under optimal conditions the speed with which the channel can move data from the core of one machine to the core of the other, is determined by the maximum rate that the PDP-10 processor can service requests on the I/O bus. This rate is 200,000 36-bit words per second. Thus it is possibly to perform an entire core load on the PDP-11 with its 120K of 16-bit words in .4 seconds using a format that generates 2 PDP-11 words from a single PDP-10 word.

The channel hardware physically consists of two racks of printed circuit cards, one mounted in each machine. The two racks are connected by 40 coaxial lines, 16 for data, 16 for control, and 8 spare.

If a significant event occurs, the channel hardware informs each processor by setting the appropriate bit in their respective channel status words. If the appropriate interrupt enable bit is also set, which is the case for normal operation, then an interrupt will occur on that processor. From now on we will assume that all the interrupt bits are set. The channel design philosophy emphasize seven major points: (1) to achieve high data transfer rates the channel makes use of the direct memory access "DMA" capabilities of both machines. On the PDP-11 side the channel hardware becomes master of the unibus and reads from, and writes to the memory, which is just another device on the unibus, without interrupting the program executing on the PDP-11 processor. Thus the program on the PDP-11 is slowed down only by the competition between the channel and the CPU for the unibus cycles. At worst this would slow the program down by about 30%. On the PDP-10 side the channel hardware makes use of the new KI-10 "data type"

interrupts to force the PDP-10 processor to execute a partial I/O instruction to transfer a word of data between the channel on the I/O bus and the memory. This of course, transparent to the execution of PDP-10 programs, however it is also transparent to the monitor as well. In other words no monitor interrupt routine is required for this type of data transfer. One of the major advantages of using this new KI-10 interrupt is that data is loaded into mapped memory rather than into absolute memory addresses.

(2) For reliability, all data transmissions use full handshaking synchronization. If the PDP-11 is sending data to the PDP-10 for example, then the PDP-11 will not send another data word until the PDP-10 has read the first data word. In other words both machines can service the channel as slowly as they like without jeopardizing the reliability of the data. This synchronization is accomplished using two "done" bits, one for each machine. The exact sequence of events to transfer a data word from machine X to machine Y is as follows: 1) the done bit on machine X goes high, this causes a DMA interrupt, 2) a word is retrieved from the memory of machine X and placed in the channel, 3) machine Y receives the data word, 4) machine X clears its done bit and sets the done bit on machine Y. This causes an interrupt on machine Y, 5) Y now removes the data word from the channel and places it into its memory, 6) Y clears its done bit and sets the done bit on machine X, and the cycle repeats.

(3) The channel is a simplex bidirectional communication link. Thus a data block can be transferred in only one direction at a time. The simplex implementation was chosen over a full duplex realization because it requires half the hardware. The problem of finding the direction of data

flow in the next communication is determined by the channel arbitration logic. Each processor may, by setting a bit, make a request to use the channel to write to the other machine. If both processors make such requests simultaneously then the use of the channel is granted by the arbitration logic to one processor or the other.

(4) All communication over the channel is done by mutual consent. In other words both processors must agree by setting their respective "device active" bits to a data transfer before it can occur. It is impossible for one processor to force a data transfer upon the other via the channel hardware only. This feature protects the PDP-10 timesharing system from an erroneous or poorly debugged program on the PDP-11 causing a transfer that might corrupt the PDP-10's memory. Similarly it protects users on the PDP-11 from PDP-10 users who might attempt to access the PDP-11 remotely without checking if this action might cause some conflict.

(5) Every attempt has been made to make each half of the channel operationally symmetric. Thus the same sequence of events that would invoke a read block operation on the PDP-11, for example, would perform similarly on the PDP-10, although the bits involved may be in different locations in the respective status words. This concept fits well with design philosophies 3), 4). In addition, it makes the channel easier to program and easier to use since a description of how the channel words need only be given for the PDP-11 side, say, the operational description for the PDP-10 side being identical.

(6) Upon a request from the PDP-11, the channel also has the capability of bootstrapping the PDP-11 from the PDP-10. To make a bootstrap

request the PDP-11 processor executes the fourth status word of the channel. At this location is the instruction "CLR-(PC)" which when executed out of read only memory is a one instruction infinite loop. When the channel hardware senses the execution of this instruction at this particular location the "PDP-11 bootstrap request" bit is set on the PDP-10 side. This causes an interrupt on the PDP-10 side. In addition, the PDP-11 side of the channel hardware is primed to receive an arbitrarily long block of data from the PDP-10 starting at memory location zero, and in a predefined format. After the PDP-10 has sent the bootstrap program to the PDP-11 the PDP-10 sets a write only bit on its side and the "CLR-(PC)" instruction, which the PDP-11 has been executing repeatedly up till now, is changed by one bit to the instruction "CLR PC" which causes the location counter to jump to location zero and start execution of the newly loaded bootstrap program. Since only one bit is changed in the "CLR-(PC)" instruction it need not be synchronized to the execution cycle of the PDP-11 CPU. If 2 bits had to be changed, say from 00 to 11, then there would be the problem of the CPU executing the instruction at just the wrong instant, like when the bits were 01 or 10.

(7) The difference in word size, is perhaps the main obstacle in designing an efficient communication channel between the PDP-10 and the PDP-11. The PDP-11 has a 16-bit word while the PDP-10 has 36 bits per word. In any ZZZZ processor communication link it is desirable for a processor to send and receive full words of data. Most programs normally expect data to come in full word chunks. Unfortunately, the first single bidirectional mapping that would pair full 16-bit words to full 36-bit words would map 9 PDP-11 words to 4 PDP-10 words. Such a mapping is useless since considerable

effort would be required to unpack the data bits into a useful form. For this reason it was necessary to use two bidirectional mapping schemes. This is accomplished by a section of the channel hardware called the mapping unit. In the first scheme (16-bit mode) full PDP-11 words are mapped to and from the low order 16 bits of the 2 halfwords of a single PDP-10 word. The first PDP-11 word is constructed out of the low order halfword, bits 20 through 35. The second PDP-11 word is constructed from bits 2 through 17. Bits 0,1,18,19 of the PDP-10 word are normally thrown away, however they are available to unorthodox programs as the high order 4 bits of a third PDP-11 word. Thus using this first scheme the PDP-11 can send and receive full words of data. This mapping scheme is useful for transferring PDP-11 load modules and other "core image" data. It is used in all the low level software handshaking between the two machines, and in general is the best compromise in data formatting between the two processors.

In the second mapping scheme (12-bit mode) a full 36-bit PDP-10 word is mapped to and from the low order 12 bits of 3 PDP-11 words. Bits 15 through 12 of each PDP-11 word are discarded. Bits 0 through 11 of the first PDP-11 word are mapped to bits 35 through 24 of the PDP-10 word. The second PDP-11 word is mapped to bits 23 through 12, and the third is mapped to bits 11 through 0. Note that conventions for numbering the bits on the PDP-10 are opposite of those for the PDP-11. On a PDP-11 bit 15 is the most significant bit of a word, whereas on the PDP-10 bit 0 is most significant. This format was designed for the transfer of video display files, six bit ASCII, and "PDP-10 image" files that contain 36 bit words. One of the intentions here is to use the PIP program on the PDP-10 system without

modification to transfer files to the PDP-11.

All of the mappings are bidirectional. Thus while bits must be thrown away when the data is compressed, I.E. 16-bit mode PDP-10 to PDP-11, something must be done to fill these bits when the data expands, I.E. 16-bit mode PDP-11 to PDP-10. There are three possibilities, all of which are provided by the channel hardware. The extra bits can be filled with zeros, ones, or the sign can be extended.

The mapping unit can also throw words away. It is possible, for example, in the 12-bit mode to construct only two PDP-11 words out of one PDP-10 word. This would be done by skipping the third PDP-11 word. Similarly for one word to one word mapping in both modes.

Whenever a communication link maps one word of data on machine X to multiple words of data on machine Y, a boundary problem exists. If a data word is automatically read from the memory of machine X, expanded, and loaded into K consecutive memory locations in machine Y, by the hardware of the communication link, then it would be impossible fro machine Y to have buffers of size N filled exactly with data unless N is an integer multiple of K. Using the 12-bit mode K = 3 and N = 256, which is standard for DEC's DOS (Disc Operating System) it would be impossible to completely fill the buffer. The buffer would have to be filled either one short of two words over. Both of these alternatives are unacceptable. In order to avoid this problem the communication channel has two internal conditions bits that indicate how many PDP-11 words will be formed out of the first PDP-10 word. In the above example, the initial condition bits would be set up so that one PDP-11 word would be constructed out of the first PDP-10 word. This is done

by throwing away first two out of three PDP-11 words and the remaining 85 PDP-10 words would each form three PDP-11 words, to fill exactly, the 256 word buffer.

These two bits along with the mode bit define the state of the mapping unit. Since these bits are not affected by an interrupt caused by the channel hardware, it is possible, when one machine receives a "buffer full" interrupt to redirect the channel hardware to continue to load data at a different core location even though other machine has not finished with its buffer. Thus the PDP-11, as well as the PDP-10, can switch data buffers in the middle of a data transfer transparently to the other side without loss of continuity.

The channel hardware has the capability of accessing in sequentially increasing or sequentially decreasing order, the memory of each machine. Not only does this allow for the reformatting of data in an array, but in fact, one of the more useful modes is to access the memory of both machines backwards. This is the only way that 12-bit data in a form amenable to the PDP-10 byte instructions can be transferred to the PDP-11 in sequentially ascending memory address locations. This being a useful format for data processing on the PDP-11.

A word should be said about the software concerned with the operation of the communication channel. After being assigned the use of the channel by their respective monitors, programs on the PDP-10 and PDP-11 may communicate in one of three ways. 1) The PDP-10 program will issue write block command and the PDP-11 program will reciprocate by issuing a read block command. 2) Reversing the roles, the PDP-11 program will issue the write block command

and the PDP-10 program will reciprocate. 3) They may both issue read and write block commands. The read and write block commands are received by their respective monitors and passed to a software package within the monitor called the channel driver. The channel driver is responsible for establishing communication with a similar package on the other side, and executing the low level dialog with the other driver before the actual data can be sent. The driver must also be prepared to handle any error conditions that might arise. The PDP-10 driver may also provide the bootstrapping service for the PDP-11.

A driver will typically attempt to establish communications with the other driver, the first time it is called by the monitor. To establish communications the driver must set a "device active" bit. Only after both "device active" bits are set will the channel become active. This event is indicated in the channel status words of each machine, and can cause respective interrupts if the appropriate interrupt enable bit is set. Since the channel is simplex only one driver can talk at a time, the other must listen. Due to the symmetry of the implementation, both drivers will typically want to talk at the same time. This conflict is resolved by the channel arbitration logic. Each driver must place a "request to transmit" command to the channel arbitration logic. If the other driver has not made a similar request then the first driver is granted the right to use the simplex channel to transmit information to the other driver. If both drivers make the request simultaneously then the transmit rights are given to one driver or the other and not both. The decision of the arbitration is indicated by bits in the channel status words, and a respective transmit and receive interrupts are

caused. After a driver is finished transmitting it clears its "request to transmit" bit and waits for a reply.

These short transmission typically one or two PDP-10 words volley back and forth until the two drivers decide, which direction the data blocks will go first, what format the data will be in i.e. 16 or 12 bit mode, and the number of data words in the block. After the arbitration logic grants the use of the channel to the transmitter of the data block, each side receives the appropriate transmit/receive interrupt. At this point rather than sending over one or two words explicitly, each side loads an address and word count register and sets a DMA enable bit. When both DMA enable bits are set, the data block is transferred automatically form one machine to the other without further intervention by the software. As each data word is read/written to the memory of each machine the respective word count and address registers are incremented/decremented. It should be noted that the address and word count refer to the words on that machine. Thus if the mapping is set up such that one PDP-10 word is mapped to three PDP-11 words then for each time the PDP-10 word count register is incremented the PDP-11 word count register will be incremented by three. The DMA transfer is stopped when either of the word count registers equal zero. Normally they will both hit zero at the same time. If not the machine with the word count equal to zero can reset the word count and the data transfer will continue without loss of continuity. After the machine transmitting the data block is finished, it releases its transmitting rights thus interrupting the other side, and this event is used to indicate that the transfer is complete. It should be noted that normally both of these interrupts occur at the same time. If something

has gone wrong, however, they will not be coincident. To conclude the transaction the data receiver will return another short message indicating to the data transmitter that all the data was received correctly.

Before the driver returns to the monitor for the last time it should clear its "device active" bit and thus indicate to the other driver that it may no longer exist in core. The "device active" bits are cleared by a "reset" instruction on both machines, or by power up conditions. In general, nothing can happen unless both "device active" bits are set.

The process of bootstrapping the PDP-11 is identical to the description for loading a data block, except that everything is done automatically by the channel hardware on the PDP-11 side rather than by software. Unlike the software the hardware is incapable of handling error conditions that may occur during the bootstrapping operation. If the PDP-11 is bootstrapped in this manner, it is the only time that the PDP-11 is at the mercy of the PDP-10. At any other time, if sufficiently comprehensive error handling routines exist in the PDP-11 channel driving software, it is impossible for the PDP-10 to take over control of the PDP-11. Of course if the PDP-11 software is cooperative then anything is possible. It is this property, that the PDP-11 is a slave only when it wants to be, that forbids there being multiple possibly incompatible masters for one slave. The result in such a case is chaos.

Both the PDP-11 and the PDP-10 are capable of supporting their own monitors and operating independently of each other. The communication channel was designed to maintain this property and thus it is impossible to "take over control" of one processor from the other via the channel hardware only. The major part of this security is provided by the fact that each

the DMA transfer of data on its side. The other aspect of security, is that the channel hardware prevents one processor from "hanging" the other. In other words it is impossible for one processor to prevent the other from doing any useful work. This type of interference can take on several forms. If during a data transfer either side hangs up the hook by clearing the "device active" bit or by releasing transmission privileges then the other side is informed of this event with an interrupt and the appropriate status bit set. Basically if any change occurs in the state of the channel hardware then, if it is appropriate to do so, the other side is informed with an interrupt and a bit set in one of the status words indicating what condition caused the interrupt. If for some reason one side has gone into an infinite loop and is continually causing interrupts, faster than the other side can service them (an interrupt is serviced when tthe bit indicating the interrupt type in the status word is cleared) then a loss of information interrupt occurs. At this point the appropriate thing for the processor to do is to completely recycle the channel hardware as though it had just been started up. Each side of the channel has one DMA enable bit and two interrupt enable bits. The first enables the word count equals zero condition. The other enables all of the change of state interrupts. A change of state interrupt occurs whenever a significant change occurs in the state of the channel hardware i.e. when the channel becomes active for one side or the other gains transmit privileges or the other side becomes inactive, or a loss of information occurs. The explicit state of the channel can also be read by both side. However, this information is less important.

In conclusion, it is perhaps only fair to note that the construction of the communication channel has not been without its growing pains. Most note worthy of these is the original design of the high speed coax transmitter/ receiver cards, which developed internal oscillations, probably due to capacitive cross coupling between circuits. It is now recognized that the channel hardware was overdesigned in several respects, most notably speed. It turns out that almost all the programs that use the channel are computaton bound. Due to the care in documentation taken during the design stage the debugging of the channel has been reasonably straight forward. The channel has now been in limited operation for the last two months now, quite to the satisfaction of those users concerned.

3.3. PDP-11 System Software

3.3.1. Introduction

To aid in software development and provide ease of operation the M and M system has been written. This monitor while being specifically designed for the PDP-11 system at CSL has many features that could possibly recommend it to a wider user group. A general description of the system's facilities will follow.

The M and M system is designed to run on a PDP-11/40 with memory management option, a system console, a booting device (preferably dectape or disk), and at least 48K of main memory. The system itself in the current configuration requires 28K and part of a 4K buffer area (the rest will be used for user request ED buffers).

3.3.2. 124K Memory Utilization

The M and M (Much Memory) system has been designed primarily to

make use of a hardware system with much more memory than is easily accessed by a single user partition. Whenever possible system data structures have been built in memory partitions that differ for the user's (either in the kernal memory space or in transient pages). Also several monitor calls have been provided to allow user controlled access of extended memory (memory beyond the user's 32K address space).

At this point it would be profitable to briefly discuss the PDP-11 paging hardware so that the methods employed in system paging will be more clearly understood.

The PDP-11/40 memory management option consists of 16 page descriptor registers (PDRS), 16 page address registers (PARS), and 2 status registers (SSRO and SSR2). Of the 16 PDRS 8 are used for user addressing (UPDO-UPD7) and 8 are for kernels addressing (XPDO-XPO7). Similarly 8 of the PARS are for user space (UPAO-UPA7) and 8 are for kernel space (XPAO-XPA7).

The user or kernel addressing spaces are divided into 8 4K word segments. Each of these is described by a PAR-PDR pair. The first 4K (addresses 000000-017777) is referred to by PARO-PDRO. The next (020000-037777) by PAR1-PDR1 and so on.

When a virtual address (16-bit) is generated by effective address calculation, the top 3 bits are used to access one of the 8 PAR-PDR pairs in the user or kernel maps. The PDR tells the length of the page allocated in that 4K block of addresses, the segment access code, and the direction of expansion. This information is used to determine if the effective address is obtainable. If not a page fault trap occurs with information concerning

the fault returned in SSRO and SSR2. The PAR is used if the page is properly accessed as a base for calculation of an 18-bit unibus address. The low order 6 bits of the effective address are sent straight through to the unibus address lines thus selecting a byte or word in a 32 word block. The next 7 bits are added to the selected PAR to obtain the high 12 bits of the unibus address.

The processor status (PS) is used to select the proper page map set to use in memory addressing. If the high order 2 bits (bits 15-14) are 11 the user maps are used for all instructions excrpt MTPI and MFPI (move to/from previous instruction space). If they are 00 the kernel maps are used. MTPI and MFPI are used to access core locations in the address space where traps originated from. As such these instructions fetch all address operands from the address space selected by the high order 2 bits of the PS and fetch (or write) data operands from (to) the address space selected by bits 13-12 of the PS (00=kernel, 11=user). These bits are set by each interrupt transaction so in the case of the various trap instructions (EMT, trap, IOT, BPT) The address space from which the trap was fetched is set.

To change the user addressing space totally (as done during a time slice) all user PARS and PDRS must be saved in a control block and replaced with new values using the move instruction.

The M and M system uses the paging hardware in 3 basic ways. First, much of the monitor itself is not always paged. Different system modules are paged into core by request through user issued monitor calls (EMT instructions). Second, a set of monitor calls are provided to allow user tasks to request particular 4K segments to be mapped by name or by segment number (0-31), 31=device register addresses. The M and M system allocates all core using

4K multiples for ease in use of the memory mangement hardware. Third, users may elect to write applications that contain several concurrently running processes. Each process may be allocated independent or overlapping core segments (and therefore address spaces).

3.3.3. Input/Output Modularity and Debugging

There are two major categories of input/output equipment attached to the PDP-11 hardware system. These are the tty-like devices and the "DMA"-like devices. "DMA" is used to indicate non-tty rather than actual DMA(direct memory access). All DMA devices are also "DMA" and for software purposes both are the same (in that they both require driver modules). DMA will be used to refer to both from now forward.

All tty-like devices are serviced through a re-entrant handler embedded in the system file-structure programs. The DMA-like devices each are serviced by independent device driver programs. The driver provides the software interface from the device hardware to the DMA device independent file handlers. To extend monitor I/O service to new classes of devices (currently RK11 disks, CSL communications channel, and dectape are supported), new device drivers need only be written.

The system file structure programs provide device independence for sequential input/output among all devices (tty-like and DMA). Only DMA devices may support direct access and multiple channel I/O (more than one input and output channel open at once). Sequential files use monitor controlled buffer areas created in extended memory and accessed on input/output calls using an external page (a page reserved in the kernel address space for various operations in extended core).

A debugging package is provided for system and user task debugging. This set of routines provides trace back information after failures, trace and breakpoint facilities, core manipulation facilities, and system status checking. All operations of the debugging modules are designed for minimum interaction with the routines being debugged.

All communications to the system is accomplished by the EMT trap instruction. This instruction causes an interrupt sequence which enters the system EMT handler. From there the appropriate system module is paged into addressable memory and transferred to. On return the system maps are restored.

3.3.4. Medium Zero Program

This system program is used to initialize direct access media for M and M system file structured input/output. The initialization parameters are obtained from the appropriate device block in the monitor (using the device characteristics EMT).

For directory structured devices this program performs a bad block analysis and then writes a medium directory with bad blocks flagged.

3.3.5. Multitasking and Real Time Trapping

One user address space is initialized at monitor start up time. This space is used to run the first (root) user task. This task can then start other tasks and start system time slicing.

The algorithm used to choose the next task when context switching is quite simple. Any task not in wait state flagged active is a candidate. The one actually chosen will be the one having the highest run priority (set at task start time). If several tasks have the same priority a

round robin scheme is used to choose the next to run. That is, the same task will not run twice in a row.

Context swapping consists of changing user maps, user registers, user information in the kernel (such as project-programmer number, map saves, etc.), and kernel stacks. Each user task is allocated a job table entry for holding the various task specific information and a system stack. During an entire user's time slice his system stack is used for all transactions (including interrupts).

Timesharing (context swapping) may voluntarily be turned off by any user task (thereby freezing that task in the user maps). Also on any fault condition timesharing is frozen. If the fault may be recovered by the operator and the system is continued timesharing is also continued. If the fault is terminal the system will have to be reinitialized following any debugging interogation by the operator.

User tasks may request that segments be mapped into kernel address space for extremely fast interrupt processing. If slower speed response is tolerable the user may set up an interrupt routine in his own area. On interrupt (or timer calls) the user's routine will be paged into the user address space and transferred to. This allows highly flexible real-time interrupt processing in any user task.

3.4. Flight Equations

The following is a presentation of the equations used in the aerodynamic simulation. An explanation of the terms involved follows the presentation. The order of presentation is not necessarily the correct order for programming the equations.

1. LIFT COEFFICIENT:

```
CL=CLA*ALPH+CLDE*DELE+CLAD*ALDOT+CLQ*(Q*CBAR)/(2.0*V)-CLO
```

2. DRAG COEFFICIENT:

```
CD=CDO+(CL*CL)/(PI*EO*AR)+CDDE*DELE
```

3. LONGITUDINAL FORCE COEFFICIENT:

```
CX =CL*SALPH-CD*CALPH
```

4. VERTICAL FORCE COEFFICIENT:

```
CZ=-CL*CALPH-CD*SALPH
```

5. PITCHING MOMENT COEFFICIENT:

```
CM=CMO+CMA*ALPH+CMDE*DELE+CMQ*Q*CBAR/
(2.0*V)+CL*(CG-CGREF)+CMAD*ALDOT
```

6. ROLLING MOMENT COEFFICIENT:

```
CSL=CSLDA*DELA+CSLP*P*B/(2.0*V)
```

7. AERODYNAMIC FORCES AND MOMENTS
   (L=ROLLING MOMENT, M=PITCHING MOMENT):

```
XAERO=CX*QI*S
ZAERO=CZ*QI*S
LAERO=CSL*QI*S*B
MAERO=CM*QI*S*B
ACCELERATION:
```

8. ALONG X AXIS:

```
UDOT=W*Q+(XAERO+XTRST)/MASS-GZ*(-STHET)
```

9. ALONG Z AXIS:

```
WDOT=U*Q+ZAERO/MASS-GZ*CPHI*CTHET
```

10. OF PITCH RATE:

    THDOT=Q*CPHI

11. OF ROLL RATE:

    PHDOT=P

The aircraft rates are integrals of the above accelerations with the following additions.

1. ROLLING RATE:

   PDOT=LAERO/MOINX

2. PITCHING RATE:

   ODOT=MAERO/MOINY

The pitching and rolling moments are the integrals of the above rates. This incomplete treatment of the equations of flight is rounded out with the following equations.

1. AIRCRAFT VELOCITY:

   V=SQRT(U*U+W*W)

2. RATE OF CLIMB:

   ROC=W

The altitude is a function of some initial altitude and the integral of the rate of climb. The heading is a function of roll angle.

EXPLANATION OF TERMS:

1.	CLA	LIFT CURVE SLOPE	PER RADIAN
2.	ALPH	ANGLE OF ATTACK	RADIAN
3.	CLDE	ELEVATOR LIFT CURVE SLOPE	PER RADIAN
4.	DELE	ELEVATOR DEFLECTION	RADIANS
5.	ALDOT	ANGLE OF ATTACK RATE	RAD/SEC
6.	CLAD	ANGLE OF ATTACK RATE LIFT CURVE SLOPE	PER RADIAN

7.	CSLQ	PITCHING MOMENT PITCH RATE DERIVATIVE	PER RADIAN
8.	Q	ANGULAR RATE ABOUT THE Y AXIS	RAD/SEC
9.	CBAR	MEAN AERODYNAMIC CHORD	FEET
10.	CLO	LIFT COEFFICIENT AT NO ELEVATOR DEFLECTION	---
11.	CDO	MINIMUM DRAG COEFFICIENT	---
12.	CDI	INDUCED DRAG COEFFICIENT	---
13.	CDDE	ELEVATOR DEFLECTION DRAG	PER RADIAN
14.	CMA	PITCHING MOMENT COEFFICIENT SLOPE	PER RADIAN
15.	CMQ	PITCHING MOMENT RATE DERIVATIVE	PER RADIAN
16.	CMO	PITCHING MOMENT COEFFICIENT AT ZERO LIFT	---
17.	CMDE	ELEVATOR DEFLECTION PITCHING MOMENT COEFFICIENT	PER RADIAN
18.	CMAD	ANGLE OF ATTACK RATE PITCHING MOMENT DERIVATIVE	PER RADIAN
19.	CG	CENTER OF GRAVITY LOCATION	% CHORD
20.	CGREF	AERO DATA REFERENCE	% CHORD
21.	CSLDA	AILERON DEFLECTION ROLLING MOMENT DERIVATIVE	PER RADIAN
22.	DELA	AILERON DEFLECTION	RADIANS
23.	CSLP	ROLLING MOMENT ROLL RATE DERIVATIVE	PER RADIAN
24.	P	ANGULAR RATE ABOUT THE X AXIS	RAD/SEC
25.	B	WING SPAN	FEET
26.	XTRSTL	THRUST,LEFT ENGINE	POUNDS,FORCE
27.	XTRSTR	THRUST,RIGHT ENGINE	POUNDS,FORCE
28.	MASS	AIRCRAFT MASS	SLUGS

29.	XTRST	TOTAL THRUST, BOTH ENGINES	POUNDS,FORCE
30.	QI	DYNAMIC PRESSURE	POUNDS PER SQUARE FOOT
31.	S	WING AREA	SQUARE FEET
32.	GZ	ACCELERATION OF GRAVITY	FEET/SEC/SEC
33.	W	VELOCITY ALONG Z AXIS	FEET/SEC
34.	U	VELOCITY ALONG X AXIS	FEET/SEC
35.	PHI	BANK ANGLE	RADIANS
36.	MOINX	MOMENT OF INERTIA ABOUT X AXIS	SLUG-SQUARE FEET
37.	MOINY	MOMENT OF INERTIA ABOUT Y AXIS	SLUG-SQUARE FEET
38.	THETA	PITCH ANGLE	RADIANS
39.	()DOT	DERIVATIVE OF ()	---
40.	C( )	COSINE OF ()	---
41.	S( )	SINE OF ( )	---
42.	SQRT	SQUARE ROOT	---
43.	AR	ASPECT RATIO	---
44.	EO	OSWALD'S EFFICIENCY FACTOR	---

4. ADVANCED CONCEPTS

4.1. Planning and Execution in Incompletely Specified Environments

4.1.1. Introduction

In conventional computer applications, data is input and manipulated according to predefined plans specified by the programmer. Programs are written to provide solutions to problems for which algorithms are known. While the numerical data may be changed easily, altering the goal of the program may require extensive modification. This type of approach may be unacceptable in systems where the exact problem specification is not known at the time of programming.

In order to create programs which can solve a wider variety of problems, much time has been spent constructing systems which attempt to make the computer "understand" the subjects with which it is dealing. Many of the systems are planners, i.e., programs which take a goal as an input and generate a plan which can be executed, at which time the given task will have been satisfied.

Most of the existing high level planners such as STRIPS[4] and PLANNER[6,14] will report a success only when a detailed plan has been developed. These planners have primarily been applied to simple domains in which all relevant aspects concerning the state of the world are known to the planner. In these domains, nothing can change without the execution of a system initiated action.

Many of the planning systems are based upon the idea that a problem may be divided into a series of subgoals (or preconditions).

Any one of a number of techniques or operators can be employed in order to try to satisfy the subgoal. A sequence of correct operators would determine the solution. However, much of the deduction may depend upon the presence of certain data in the database. If this data were not known at the time of planning, then an entire section could fail. In many cases information may be missing because of incomplete modeling of the world due to the domains complexity. But in other cases, the overall concept may have been modeled, but the specific piece of datum may not be "known" to the planner, much as a person may not know what is going on in the next room.

In order to increase the complexity of the problems and domains that can be accommodated, it is necessary to extend the capabilities of the deduction languages and systems in the following areas:

The deduction mechanism must have the capability to construct plans in a dynamic environment. By dynamic, it is meant that movements of objects or changes in the world can occur without being merely consequences of system actions. In this type of situation, it may be futile to formulate a detailed plan based upon specific data when a dynamic alteration of this data may totally invalidate the remainder of the plan. Planning for all of the possible alternatives would, in most cases, be unfeasible due to the large number of future states possible. One alternative would be to have a planner which would "know" that it exists in the real world. The planner would have the ability to vary the complexity of the plans generated according to the situation. This would lead to a case where there would be no necessary distinction between the planning and execution phases. In certain cases, the plans which would be generated would be of a more

general nature, reflecting an outline of the important steps and tasks to be done. As the execution progresses, new information would be received and added to the database. This would allow more details of the plan to be computed as execution continued.

A deductive mechanism should be able to operate in an environment in which there is a lack of information. This would correspond to the real world situation where a human being has to make an intelligent evaluation when some of the facts are missing. This capability has to be attained before the dynamic planning and execution can occur. This is because while it may be recognized that a certain aspect of the world may be expected to exhibit dynamic action, it may not be known what the value would be when needed. Planning may have to continue without the definite information. The planner must have the ability to gather new information. Among the possible ways in which this could be accomplished are: have the system develop a question or allow the mechanism to seek out information by inspecting the environment using any sensory equipment available. If it is realized that while certain possibly relevant information is not known during the periods of initial planning, it may be possible to plan if it is realized that the information is to become available at some future time. In this case, it may be necessary to analyze certain possible future states of the world. For this, there must be a mechanism for storing global knowledge and models concerning this unknown information. It may also be desirable to incorporate probabilities and costs into the deductive and decision making procedures.

4.1.2. Related Research

Research concerning the application of general deduction mechanisms to real world problems which are dynamic in nature and/or are incompletely specified have been extremely limited. It appears that many of the existing systems are incapable of being extended to handle these types of problems without extensive modifications.

PLANNER[6,14] allows strategies and relationships to be expressed as procedures called theorems. These theorems are executed in order to try to satisfy the problem which consists of a series of goals. The control stucture is based upon a depth first search or backtracking. It appears that it would be difficult to express the idea that certain facts may not be known at a given time. PLANNER understands only one type of failure, that being when a goal cannot be satisfied. If, however, a goal is not satisfied not because it is "wrong" but because some of the necessary data are missing, then a different type of failure has occurred, a type which PLANNER cannot understand. When dealing with this "unspecified information", it is necessary to maintain several alternatives from which one may be chosen. Storing this type of information is very difficult in PLANNER.

When evaluating a theorem, PLANNER treats each of its steps or subgoals as equal. During the planning, all of these subgoals are of equal importance. If one fails, the theorem fails. But, it appears that in reality, subgoals have different levels of importance. This means that more time should be spent in order to satisfy a key subgoal than a relatively minor one. PLANNER's depth first control structure would not allow the

consideration of all major subgoals first.

Many of the philosophies in PLANNER are also contained in QA4[10]. The context mechanism which is available in QA4 would allow the storage of alternative plans resulting from different possible values of unspecified information. QA4's limited effectiveness, as with PLANNER, arises from the backtracking philosophy which is embedded in both the systems. The introduction of new types of failures makes backtracking an undesirable search technique. As in PLANNER, the inflexibility of the recommendation lists means that once a sequence of theorems is formed, it cannot be altered or edited. This appears to be inappropriate when desiring to alter control as new information is determined.

CONNIVER's[7,13] main advantages over PLANNER and QA4 are freedom from compulsory backtracking, the inclusion of a context mechanism, and flexible possibilities lists. The possibilities list, which specifies the next procedure to be tried, can be inspected or edited at any time. The control structure is based upon the frame concept[1] which allows a total deduction environment to be maintained and continued. This allows great flexibility in specifying how a theorem is to be evaluated. Despite its advantages, it appears that CONNIVER has not yet been applied in systems which require the integration of planning and execution, such as those dealing with dynamic situations in uncertain environments.

Much of the work which has been done concerning the problems found in executing and planning have been outgrowths and extensions of the STRIPS[2,3,4,5] system. STRIPS is used to generate a plan which could be solved through the application of a sequence of operators. An operator can

be executed when all of its preconditions have been satisfied. The PLANNEX[6] system takes a complete STRIPS plan and monitors its execution. Using this system, actions may be deleted from the plan if it is determined that their consequences are not needed. It can also recognize if necessary initial conditions are absent, which would lead to a replan mode. It is also possible to take solutions which have been generated and generalize them. These MACROPS[5] are saved and can be used to satisfy future tasks or subtasks. STRIPS only succeeds when a complete plan has been generated. The presence of unspecified information would in most cases lead to a failure. STRIPS would respond to this type of failure by searching for an alternate plan.

Recent results have demonstrated that STRIPS-like systems can be made more efficient by employing a hierarchical approach[9,11,12]. These systems have been constructed using the principle that preconditions of an operator are of varying importance and some should be examined and satisfied before others. By trying to satisfy the preconditions which are most basic or are harder to achieve first, irrelevant operators can be eliminated sooner. The preconditions are assigned a criticality or rank. The higher valued preconditions represent the tasks which must be satisfied first. So in ABSTRIPS[11], which plans in a robot and block domain the precondition that an object be a block has a higher rank than the precondition that the block is in a room. Both have a higher rank than a precondition which demands that a robot or manipulator is also in the room. When a problem specification is received, the criticality is set to a maximum value (which would contain all predicates representing unchangable information). Preconditions with criticality below this value are initially ignored. A plan is constructed using whatever techniques are appropriate to the system and domain. The plans

produced will satisfy all of the final conditions, but the operators specified will only be satisfied through their most critical preconditions. As the criticality is lowered, new preconditions are introduced for the already specified operators. As these preconditions are satisfied, new operators are introduced forming a more detailed plan. When the criticality has been set to its lowest value and all preconditions have been satisfied, a complete, detailed plan will have been constructed. While this type of planning has proven to be more efficient that STRIPS, of more interest are the types of plans which are generated. Some of the high-criticality plans have many of the desired attributes of a partial plan outline. The plans do not contain every necessary detail, but rather only the major steps which must occur. These approaches have not been used to satisfy problems in domains which are dynamic or incompletely specified. In [8] Minsky describes a framework for a representation of knowledge which would permit the inclusion of situation dependent default values. The scope of the world model which is considered at any time is a function of the present environment.

4.1.3. Planning in an Incompletely Specified and/or Dynamic Domain

Systems which are to operate in an incompletely specified, real world environment must of necessity operate differently than the existing planners. There must be a realization that knowledge concerning some of the relevant portions of the world may be unavailable during some of the stages of planning. This may be because the unknown portion of the world is outside of the system's monitoring capability and/or may be changing as time progresses. Because of this, the planner must realize that in many cases it is not feasible, if not futile, to insist upon construction of a completely detailed plan before the initiation of execution. The system must have the knowledge that missing

information may be obtained in various ways, such as through observation or questioning. The system may have to plan around some of the missing information by making reasonable assumptions.

When a problem is specified to a planning or execution system, it may be done in several ways. The most common method is to specify aspects of the world which must exist after the plan has been executed. This is generally accomplished by specifying a goal state. It may also be desirable to specify possible intermediate states which may have to be satisfied in a certain order. Most of the existing planners place little emphasis on how the goal state is to be achieved. There is little or no concern about whether the plan which has been generated is optimal or near-optimal according to any criteria. However, in more realistic situations, people strive for a more efficient plan even though their analysis may not include a formal statement of what is best. A planner should also be able to accept a statement of what criteria should be used.

In these types of problem specifications, there are really only three general methods which can be used to insure that a portion of the goal is satisfied. First, the goal could already be true in the world and represented in the system's world model. Second, the goal could be true in the world model but not explicitly represented in the system's model and it could be deduced that the goal is a logical consequence of available information in the model. Third, it may be that the goal is not true in either the model or the world but it is possible to perform actions which will alter the world in such a manner that the goal will be satisfied. The existing general purpose planners satisfy goals using these general

techniques.

But all of the situations above are predicated on the concept that all relevant information is directly known or could be deduced. But these are clearly not realistic assumptions. The world model which the system maintains could be deficient in many ways. Some information could be missing due to the necessary simplifications which must occur when modeling a complicated domain. However, if some aspect of the world were expected to be important for planning, it would surely be represented. And some of the information could be missing because it is just not known, no matter how relevant it may be. The latter case is of major interest because this type of unspecification occurs in realistic problems when a portion of the world is beyond monitoring capability or when dynamic situations alter previously known values.

One major problem is how to recognize this type of missing information, which will be referred to as "unknown" information. When a precondition to an operator is encountered, it is imperative to know whether it belongs to a previously mentioned category or is unknown information. No matter how complete the model, certain information may be absent if the system is solely responsible for collecting and storing the information. But by using semantic knowledge about the world, it may be possible to determine something about how to satisfy the preconditions. The initial attempt to satisfy a precondition involves examining the database in order to see if the precondition is satisfied because it is already true. As soon as it is determined that the needed fact is not in the database, the system checks to determine if the concept is unspecified.

In many cases it is possible to determine that a precondition is not satisfied by inspecting the database for contradictory information. If this is the case, then the system can conclude that the information is specified and that some action is needed. Sometimes, however, sufficient information is not available to make the determination and the system may have to postpone the decision. Initially, research concerning unknown information will be confined to predicates whose restrictions are limited. In these cases the alternate values and contradictions can be expressed in a fairly straightforward manner.

When the database is being referenced, it is not enough to find a specific fact represented. The dynamic properties of the domain have to be considered. Some of the attributes may change dynamically in a random or predetermined manner. This would affect the confidence in the truth or falsity of a fact.

In order to ascertain the value for an item of unknown information, it is necessary to activate some type of input. This may include any sensory device available, such as a camera for observation. It may also take the form of a response to a question. In any case, the system must know the appropriate methods available. It must also be aware of when types of information can be obtained.

As has been stated, a planned solution to a task is a sequence of actions whose execution would alter the world from an initial state to a goal state. The proper actions are determined by evaluating operators. The form of the operators is shown in Figure 12. The operators represent allowable actions in a domain. They are STRIPS-like in representation but

```
(TO task
    (ACTION action)
        (PRECONDITIONS
            (criticalityp1 (condition1))
            (criticalityp2 (condition2))
                .
                .
            (criticalitypn (conditionn)))
        (DELETION
            (criticalityd1 (deletion1))
            (criticalityd2 (deletion2))
                .
                .
            (criticalitydi (deletioni)))
        (ADDITION
            (criticalitya1 (addition1))
            (criticalitya2 (addition2))
                .
                .
            (criticalityaj (additionj)))
```

Figure 12

have direct counterparts in PLANNER and CONNIVER. Each operator has a set of preconditions which must be satisfied before the action can be executed (either real-time or during planning). The operators have ADDITION and DELETION lists. These represent the aspects of the world which are expected to be altered when the action is executed. These are only used to update the system's world model. The system is responsible for making any observations necessary to insure that the changes in the world correspond to the expected changes.

Each of the preconditions, additions and deletions has a number associated with it. This is the criticality of a predicate. There is an upper limit for the criticality in a domain. This is for concepts which cannot be altered by any system action. The concepts represented by predicates with lower criticalities can be changed. The criticalities roughly represent the order in which the preconditions must be satisfied. If there are two preconditions with different criticalities, the one with the highest criticality is satisfied first; the lower precondition will be satisfiable in some manner.

So, when considering an operator, the criticality is set to some value. Any precondition with criticality below this is not considered at that time. If all of the pertinent preconditions have been satisfied when the criticality is 'n', it is said that the operator is satisfied through criticality n. If all of the operators in a plan are satisfied through criticality 1, the a complete, detailed plan should have been generated.

A significant research area concerns how to satisfy the prerequisites and the development of a control structure which would facilitate the

efficient construction of intelligent plans. The conventional methods for satisfying the prerequisites have been described above. Research is being conducted concerning the possibility of considering a prerequisite to be satisfied by "assumption" at certain levels of criticality and stages of planning. In these cases, a precondition may be assumed to be satisfied (or satisfiable) at an early planning stage. The system must be aware of the assumptions being made and have reasons for these actions. To date, several classes of assumptions and their reasons have been formulated. The most basic is a low-criticality precondition. In most cases it is possible to assume that a precondition with a criticality below some cutoff can be satisfied. This is because this type of precondition was to be constructed as an easily done detail. The criticality cutoff could be determined by the stage of planning, the domain used as well as the system's knowledge of what tasks could always be accomplished.

Another type of assumption is called a logical assumption. This type of assumption originally would occur when unknown information was involved. The various possible values would have been examined. If, for each case it was determined that the precondition could be satisfied, the precondition would be assumed to be logically satisfiable. The information derived from searching all of the possibilities should some how be saved so that an assumption could be made if the proper conditions are met.

In many cases a precondition can be assumed to be satisfiable if certain relationships exist between the precondition of the operator being evaluated and the precondition of any operator used to satisfy the

original precondition. This is called a dominance assumption. An operator, OP1, is defined to dominate another operator, OP2, if all of OP2's preconditions are among the preconditions of OP1 with the same relationship restricting any variables for which values have not been determined. Now, if a precondition of OP1 may be possibly satisfied by the application of OP2, then the precondition may be assumed to be satisfiable by dominance.

In the previous two cases, the assumptions which could be made are very dependent upon the planning environment which exists when a precondition is encountered. Hopefully, this will lead to a system which has a better knowledge of what it is trying to accomplish at any given time as well as a knowledge of situation methods of dealing with the preconditions.

The last type of assumption which has thus far been considered has been called a linkage assumption. This type of assumption arises because the lack of knowledge concerning the exact order of execution of a plan satisfying part of a top level goal will cause unknown information to exist. In this case the information is known in the real world but may be altered during planning of another plan. In many cases the overall goal will be divided into subplans, each of which is developed independently. The exact order in which the plans will be executed may not be known at the time of planning. When trying to satisfy certain preconditions, a planner may examine the real-world database (as opposed to a local, planning database). But information found in the real world database may be altered by other subplans by the time the subplan is actually executed. The system must have knowledge of whether database entries are expected to change.

For those that are expected to change, the appropriate assumption may be that the precondition will be satisfiable during the execution phase.

The last type of assumption mentioned is basically restricted to attributes which are expected to change. There are some types of information which are not expected to change (even though they may). In these cases, the system may use values found in the latest real world model. But these should be noted in the plan being produced.

Another problem concerns when the system should initiate execution. This also involves the question of how detailed should the planning be. In theory, the execution could be initiated during any stage of planning. However, a more realistic approach would have the execution begin when a "reasonable" plan has been developed. In some cases the initial course of action may be so well defined (or may be the only alternative) that the system may decide to start execution before the initial planning has terminated. Observations, which are a type of execution, could be performed at any time during planning or execution if the proper conditions occur in the real world.

After the plan outlines satisfying portions of the top level goals have been generated, it is necessary to link them into a coherent plan outline. To do this, an order of execution must be determined and intermediate connecting programs must be developed. The subplans are classed according to the criticalities of the tasks which are satisfied. An attempt is made to link the highly rated subplans to the initial world. When a "shortest" linkage is found to a subplan, this subplan is assumed to be executed next. Linkage is attempted between the remaining subplans and the world model of

the most recently assumed executed subplan. This continues until all of the subplans have been linked.

Among the major problems which still must be considered for effective linking include: how to distinguish between the case when two subplans of different criticalities have the same initial environment condition and should be executed consecutively and the case when the lower criticality subplan must be executed with subplans of its own criticality to avoid making other subplans undoable; what types of searches are necessary to promote the most "efficient" linkage for the entire plan. Surely a most efficient first linkage is not enough.

It would be very desirable to have the system be aware not only of what part of the plan it is executing but also the knowledge needs and preconditions of other subplans. In the present formalation, an attempt is being made to obtain this type of performance. When observations are needed and the necessary environment does not exist, the required environment would be stored so that should the opportunity arise, the observations could be made. Certain aspects of the world could be protected as pre-conditions for future subplans, top-level goals as well as any currently active operators. If a subplan tries to undo another subplan's initial assumptions, the planner being constructed will be warned of this.

In some cases, while a plan is being executed and during an observation, some previously unknown information is acquired. This new information may lead the system to re-evaluate the manner in which the overall goal is to be satisfied. This may involve satisfying a subgoal with a different operator and/or altering the order in which subplans are executed.

For this type of performance to occur, the system must be aware of some of the important possible cases: new information leading to alternate plans, and new information leading to short cuts.

When trying to satisfy a goal, the system may be considering several possible approaches. A situation which can occur is that after planning has been successful through a certain criticality, a failure due to unknown information is encountered. In the previous cases discussed, it was possible to "assume" that the precondition could be satisfied, but this is not always the case. If the unknown precondition is found to be satisfied, the system should be able to pause and consider the new information. If a new subplan were found which was expected to be superior, the linkages may have to be reformulated. In most cases, it is not expected that this would alter the composition of other subplans. When the subplan is being reformulated, other unknown failures may still be encountered which may dissuade the system from pursuing these paths.

The possibility of a short-cut also may occur when a failure due to unknown information is encountered during planning. In this case, the operator which was being considered when the failure occurred is later found to be potentially useful in the finally constructed plan. To determine this, the system examines the failure and subplan produced and asks the questions: 1) Is the operator which was being examined when the failure occurred still potentially useful in that the change the action would have yielded was realized in some manner later on in the plan (or more precisely, did planning continue until a subplan to realize the action goal was obtained)? 2) If the failed precondition were assumed to be true,

would all of the operator's preconditions be satisfied or satisfiable to a certain criticality?

If these conditions are met, then the plan should be altered to indicate that if the precondition is found to be true, an alternate shorter subplan could replace part of the plan without affecting any of the other linkages or subplans. It is hoped that this type of planning will not only make the system more responsive to new information which is received, but will also allow the system to demonstrate a better understanding of what it is trying to do at all levels of planning and execution.

In some of the situations being considered, the cost of performing an action and probabilities of the possible values of the unknown information have been included as unspecified parameters. It is sometimes possible to judge the superiority of one plan over another with only this information.

4.1.4. Conclusions

Systems which are to be able to plan and execute solutions to real world problems must be able to plan in incompletely specified environments. The system should have the knowledge that certain relevant information may not be known at all stages of planning. The system should be able to form a plan outline indicating the "major" steps. For various situations, it should be able to assume that certain tasks are satisfiable, deferring planning until a time just before execution. The system should have the ability to initiate execution beforeplanning has terminated. As new information is received, the plan and/or flow of execution should be modifiable.

When planning for CADM, the system should be cognizant of the overall mission outline. Using knowledge of pilot and system capabilities, the plan which is constructed should be compatible with the mission objective.

REFERENCES

1. Bobrow, D., and Wegbreit, B., "A Model for Control Structures for Artificial Intelligence Programming Languages," Third International Conference on Artificial Intelligence, Stanford, California, August, 1973.

2. Fikes, R., "Failure Tests and Goals in Plans," Artificial Intelligence Group Technical Note 53, Stanford Research Institute, March, 1971.

3. Fikes, R., "Monitored Execution of Robot Plans Produced by STRIPS," Artificial Intelligence Technical Note 55, Stanford Research Institute, April, 1971.

4. Fikes, R., and N. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2, Nos. 3/4, 1971.

5. Fikes, R., Hart, P. and Nilsson, N., "Learning and Executing Generalized Robot Plans," Artificial Intelligence, Vol. 3, No. 4, 1972.

6. Hewitt, C., "Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot," Ph.D. Thesis, Department of Mathematics, Institute of Technology, 1972.

7. McDermott, D., and Sussman, G., "The CONNIVER Reference Manual," Artificial Intelligence Memo No. 259, Massachusetts Institute of Technology, May, 1972.

8. Minsky, M., "FRAME-SYSTEMS: A Framework for Representation of Knowledge," to be published, November, 1973.

9. Nilsson, N., "A Hierarchical Robot Planning and Execution System," Stanford Research Institute Report Project 1187, April, 1973.

10. Rulifson, J., Derksen, J., and Waldinger, R., "QA4: A Procedural Calculus for Intuitive Reasoning," Artificial Intelligence Technical Note 73, Stanford Research Institute, November, 1972.

11. Sacerdoti, E., "Planning in a Hierarchy of Abstraction Space," Third International Joint Conference on Artificial Intelligence, August, 1973.

12. Siklossy, L., and Dreussi, J., "An Efficient Robot Planner Which Generates Its Own Procedures," Third International Joint Conference on Artificial Intelligence, August, 1973.

13. Sussman, G., "Why Conniving is Better Than Planning," FJCC, 1972.

14. Sussman, G., Winograd, T., and Charniak, E., "MICRO-PLANNER Reference Manual," Artificial Intelligence Memo 203A, Massachusetts Institute of Technology, December, 1971.

4.2. Man-Computer Systems

4.2.1. Introduction

The design of man-computer systems presents an array of traditional problems as well as new problems that become especially significant when the computer system has some "intelligence". We have more experience with the traditional problems and some definitive answers exist. However, there is little research upon which to base solutions to the new problems. In this section of this report, we will discuss approaches to both types of problems.

4.2.2. Displays and Controls

Traditional display and control considerations are what to display, how to display it, and what form inputs should take. The question of what to display could easily be answered if we knew what information the pilot needed to make his decisions (Williams, 1947). Unfortunately, we have not completely answered this question in the almost 30 years since Williams' report. The result is that designers tend to give the decision maker (DM) any piece of information they think he might use. This can result in an information glut, increased workload, and degraded performance.

Based on a recent survey (Rouse, 1975), we can make some general design recommendations for displays and controls for man-computer systems. Display parameters of importance include luminance, contrast, regeneration rate, character size and generation method, interpolation schemes, and displays for quantitative information.

The preferred values for luminance, contrast, and regeneration rate are 50 ML, 94%, and 50 HZ, respectively. Characters should subtend 15 minutes of arc and dot matrices should be at least 10 in height with a height to width ratio of 7:5 to 3:2. Linear interpolation for the display of discrete

information can bias the statistical properties of the information which can in turn degrade DM's performance. Higher order interpolations seem to lessen this difficulty except they are much more difficult to generate in real time. For static reading of quantitative information, digital displays (counters) are preferred while for dynamic reading, hybrid displays (counters and dials) are appropriate.

The question of how to display information is also affected by the status quo in the sense that one could not expect pilots, or any other DM, to learn a completely new set of conventions, jargon, etc. Thus, display choices must allow for smooth transfer of training.

While the inputs required from DM are fairly well defined by the task, the form of the inputs is reasonably free to innovation. (Again, with the transfer of training constraint.) However, many of the numerous input devices available to the self-paced, non-moving DM are not practical for a DM who is the operator of a highly dynamic vehicle in a forced-pace situation.

There are numerous input devices for man-computer interaction. However, devices such as keyboard, light pen, rand tablet, and the SRI mouse are inappropriate since the keyboard is awkward and its data input rate is low. Light pens and styluses are easily misplaced. The mouse could easily lose its orientation in a dynamic environment. Joysticks, trackballs, and small keyboards (E.G., "touchtone") are more appropriate for the operator of a vehicle, pilots are familiar with joysticks. The 4 X 3 keyboard was used on Apollo. Thus there are precedents for these devices.

4.2.3. Task Allocation and Conflict Resolution

Perhaps the most crucial issue in the area of man-computer interaction

is the allocation of tasks and responsibility between DM and the computer. While we have a general feeling for what each should do (Licklider, 1960), reducing these principles to practice is often very difficult. Part of this difficulty stems from the computer not being able to perform many of the tasks that one would like to allocate to it. However, artificial intelligence (AI) may, in the future, produce systems capable of such performance.

The allocation of tasks between DM and AI presents three basic difficulties. While one might consider giving AI any task that it could perform acceptably, this may be inappropriate. DM should be given a task or sequence of actions that has some coherency (if only for motivational reasons). In addition, there is a possibility of underloading the DM which results in vigilance problems and degraded performance. Thus, if the human is to be part of the system, maintaining his overall performance may require that he be allocated some tasks that he performs at a level inferior to that of the computer.

Another issue that must be considered is DM's confidence in AI. If DM is to willingly give some decision making responsibility to AI, he must be confident that AI is competent and operational (has not failed). Thus, DM needs some feedback on what AI is doing. However, DM does not want to know the details since it is unlikely that DM has the time to consider such additional information. (Or, why is the computer being used in the first place?)

Further difficulties stem from what we will call "competitive intelligence". With the feasibility of DM and AI sharing responsibility for tasks, the possibility emerges that DM and AI may not agree on what to

do. In a self-paced task, this would not be too difficult to arbitrate since DM and AI could pause for a moment and debate the relative merits of each other's approach. Most likely, DM would be left with the final decision of whose approach to adopt. However, in a forced-pace, highly dynamic situation, there is no time for debate. A competitively-intelligent system might result in a much higher workload for DM since he would have to continually monitor AI. In some situations, competitive intelligence might lead to system instability. The solution is to design a "cooperatively intelligent" system.

Now we will consider the above ideas in more detail and suggest how we might solve the problems posed by these difficulties.

An initial consideration is the characterization of the decision making abilities of AI. The real world is not as neat and regular as the Laboratory and it is unrealistic to think that AI will out-perform DM in any robust set of tasks. AI has yet to perform many, if any, comprehensive human information processing and decision making tasks (Miller, 1974). Yet, AI need not perform a task better than DM to justify allocating responsibility for that task to AI. If DM has many tasks to perform, he may not be able to devote sufficient time to each task to achieve acceptable overall performance. If AI could perform tolerably well in some of the tasks, DM would be free to concentrate on a smaller set of tasks.

What tasks should AI perform? An initial and naive approach is to allocate to AI any task or subtask that it can perform at an acceptable level. This can present severe difficulties if DM and AI perform subtasks of the same task or highly intersecting tasks. The problem is that the actions which DM and AI initiate may be jointly counterproductive. In some

situations, they may be working against each other and not realize it. Or, they may realize it, but see no alternative. This phenomenon is competitive intelligence. As a perhaps unrealistic example, consider a situation where a pilot decides to dump fuel to solve an aircraft imbalance problem while AI decides to re-route fuel to solve the same problem. The result would be a loss of fuel without correcting the imbalance problem, not to mention the wasted attention of the pilot and AI.

A solution to the competitive intelligence problem might be to allocate to DM and AI sets of tasks that have little intersection (across decision makers). This may be appropriate in some situations, but it is difficult to construct sets of tasks with completely independent consequences. For example, a highly integrated system like an aircraft results in sets of tasks with many interdependencies.

If tasks are to intersect, DM and AI must know what each other is doing. DM could memorize all of AI's decision making procedures and thus be able to react appropriately to AI's decisions. However, this would probably result in increased workload and certainly more training. An alternative is to have AI know (or learn) DM's procedures and adapt its actions to DM's so as to maximize system performance. This type of system would exhibit cooperative intelligence.

Reconsidering allocation of tasks, DM and AI should each be given coherent sets of tasks with as little intersection across decision maker responsibility as possible. AI should respond in its tasks according to both the system state and what the DM is doing. This sounds rather neat yet two other issues need to be discussed before we look at the ramifications

of these ideas for CADM.

While an AI system like that described above might remove some of DM's processing load, it will not necessarily do so, since there is a non-zero probability that AI will "hang-up" or experience a hardware failure, DM still must monitor AI. If this probability is high, DM might actually spend more time monitoring AI than he would have spent if he had performed the tasks himself. A solution to this problem might be to let AI monitor itself. In other words, AI should be able to determine when it knows what it is doing, when it is having difficulty, and when it cannot handle a problem normally assigned to it. (Naturally, this approach assumes that AI has sufficient intelligence to know how intelligent it is.)

This is the principle upon which we based the green, yellow and red indicators noted in an earlier section. With these indicators, the pilot need not be concerned with the details of what AI is doing and only need divert his attention if some difficulty arises. While this approach partially handles the problem of DM knowing how AI is doing, it does depend on putting self-monitoring capability in AI and on the pilot being confident in AI.

A last issue to consider is dynamic allocation of tasks. When AI experiences difficulty or DM is facing an increased workload situation, they may want to shift responsibility for some tasks. In the first situation, when AI is having trouble, the difficulty is in smoothing transitions of responsibility. How do you give to DM all the information that AI has been using but DM has not been considering? The second situation where DM wants to shift some responsibility to AI is less difficult since AI would have

been keeping track of what DM was doing. However, defining exactly what DM wants AI to do may be somewhat difficult unless there are only a rather restrictive set of alternatives. Dynamic task allocation has not been considered in much detail during this phase of the CADM project. It will be discussed later in this report as a possible avenue of future work.

REFERENCES

1. Licklider, J. C. R., "Man-Computer Symbiosis", IEEE Transactions on Human Factors in Electronics, Vol. HFE-1, No. 1, pp. 4-11, 1960.

2. Miller, G. A., "Needed: A Better theory of cognitive organization", IEEE Transactions in Systems, Man and Cybernetics, Vol. SMC-4, No. 1, pp. 95-97, 1974.

3. Rouse, W. B., "Design of Man-Computer Interfaces for On-Line Interactive Systems", Proceedings of the IEEE, Special Issue on Interactive Computing, Vol. 63, No. 6, June 1975, in press.

4. Williams, A. C., Jr., "Preliminary Analysis of Information Required by Pilot's for Instrument Flight", unpublished report, 1947, reprinted in Aviation Research Monographs, Vol. 1, No. 1, pp. 1-17, 1971.

5. FUTURE PLANS

At this point, after successfully demonstrating a fundamental capability to apply the concepts of Artificial Intelligence to the detection and correction of single and multiple failures in a simplified airplane system, we plot the course for our future research efforts. The choices of directions were many and may have taken any of several diverse paths. But in designing a suggested approach for Phase 3, an effective and expedient balance has been maintained between the development of new ideas and physical demonstration of their feasibility.

Basically our distinct choices were whether to concentrate on filling out our present system to discover and demonstrate its ultimate capability, or whether to launch out into more long range problems that challence the state-of-the-art in Artificial Intelligence. Because of the assumed desirability of performing both research and demonstration, a coherent selection of tasks from each group have been chosen to be accomplished. Of the many available tasks, we have chosen to focus on those which exhibit or improve CADM flexibility and ability to accommodate different operating environments. The key criteria in attaining these goals are the degrees to which we are able to integrate an advanced CADM and appropriate pilot-CADM interaction so that true cooperative intelligence emerges.

Numerous first order extensions to our airplane failure model have been considered. For instance, we propose to expand our present airplane system to incorporate sensitivity to mission profile changes. This is discussed more fully in the proposed navigation system task below. The

impact on our present CADM failure model is a required dynamic reordering (according to whether the plane is landing, taking off, cruising, etc.) of our presently static hierarchy of failures, both as to criticality and to order of steps within the failure correction program. Additionally, we propose to investigate and expand the degrees of pilot-CADM interaction our present system allows. The slow interface between the operating programs and the current data base in our present model may be obscuring a potentially rich and interesting area. For instance, the "ultimate degraded mode", i.e., partical or complete CADM failure requiring (hopefully intelligent) transfer of some or all tasks to the pilot might offer some formidable problems. So also might the re-assumption of some tasks by CADM if repair or restructuring of CADM during the mission is accomplished.

Additionally, we propose to add a navigation system to our model so that a wider variety of more interesting and realistic problems may be attacked. The present airplane dynamic model seems sufficient for our tasks and we anticipate no further attempts to complicate it. Rather, we propose to expand the demonstration to include activities, situations, and data originating outside the aircraft. We propose a navigation task that requires the airplane to travel from one point on an xy plane to another with limited aircraft resources and with external obstructions such as weather storms, fixed ground obstacles, and other moving aircraft. The navigation task allows us to introduce an uncertainty factor as well as to increase the complexity of the data base. The current simulation lacks this aspect of the real world. At present all failures have known penalities and all alternative actions have known benefits. Not only is it important

to have uncertainty in the simulation, but also it is necessary to bring in the concept of mission goals. The success or failure of a CADM system in actual flight operations will be measured by whether or not it helps to achieve these mission goals. These goals go beyond the "keep the airplane flying" concept we currently use. There are, in fact, several constraining facts of flight which a decision maker must take into account (ROSCOE-MAN AS A PRECIOUS RESOURCE:-). These include:

-the performance characteristics and present condition of the aircraft;

-the presence and flight paths of traffic in the vicinity;

-the weather both local and enroute;

-the geography and topography of the terrain over which and against which the flight is made;

-any characteristics of the crew that would impose limits on the flight; and

-the body of rules that governs flight in that particular airspace.

A navigation task will provide a vehicle for presenting these constraining facts of flight to the pilot-CADM system.

A necessary prerequisite for a viable simulation at Aviation Research Laboratory will be a higher bandwidth communication link between ARL and CSL computers than now exists. A synchronous 9600 baud telephone line connection will be built for this purpose. This will require support hardware and

software on the CSL PDP-11 as well as some time on the CSL PDP-11. The current phase will see the installation of the equipments and writing of the necessary software for this link.

In the effort to complete our degraded mode operation analysis, making CADM a learning machine that attempts to discover why its choices of action were not successful, one that has some self model so that elementary automatic programming and debugging is done seems fruitful for investigation. This is a logical first step in discovering a weapon system that refines its judgements and improves with age and experience, in much the same way a pilot does. For example, the ability to abandon a slow failure correction procedure and opt for a faster one that suddenly becomes available is fundamental to efficiency and intelligence. This is relevent where any one of a number of multiple failures may be occupying (and then releasing for use by other failures) software failure correction procedures. Along similar lines, CADM as a problem solver that expands its data base and performs logical deduction and decision making will be investigated. For instance, during an attempt to correct a failure, CADM may discover that a usually effective correction procedure suddenly has no effect. CADM should then hypothesize that all air system components required by this procedure are inoperative and create a list of these components. CADM should then drop from or retain on the list components successfully or unsuccessfully used by other correction procedures until the defective component has been identified. Additionally, though it is not clear at this point that adaptive failure correction, i.e., trying to decide before beginning correction which procedure is likely to be more efficient, is more intelligent or faster than hierarchical list processing

CADM now does, this also appears to be a fertile area of investigation.

We also propose that the advanced Phase 3 CADM be able to operate effectively with some incomplete and possibly conflicting data. For instance, in addition to the weather and traffic uncertainties previously discussed, we plan to expand our airplane system to allow sensor failure and redundant sensor conflict. CADM will then be extended to cope with the situation.

Of utmost necessity during Phase 3 is the enhancement of the symbiotic relationship between the pilot and CADM. We propose to concentrate on four main efforts. First, the data displays and techniques necessary for effective interface will be investigated. The MASTER MONITOR DISPLAY may be enhanced to allow the pilot to request more detailed information about the aircraft. This additional information may be presented in the form of annotated schematics perhaps in a manner similar to Hughes (Hughes, 1974). Also we want to allow some form of direct pilot-CADM communication.

The second main effort will be that of proving the feasibility of using a model to predict pilot perceptions. This will probably require more than a single parameter per axis. It would seem that variations in the crossover model time delay would give additional information about the pilot's allocation of attention. Thus, these parameters (one for each axis) will be identified instead of assuming them fixed. We also plan to look at various identification algorithms and consider sample size, time, and accuracy tradeoffs.

A third effort will consider dynamic task allocation. This will include investigation of the human's ability to make such allocation decisions as well as how masses of information can be transferred with

the responsibility for a task. Also of interest is CADM's ability to make allocation decisions and thereby lessen the pilot's decision making load.

The fourth major effort is the rigorous experimental evaluation of program intelligence to the pilot. Subjects will perform various flight maneuvers with and without CADM detecting and correcting failures. With these experiments, we will be able to determine whether or not CADM significantly affects the overall system performance and how the effects are related to the difficulty of the pilot's task.

In conclusion, our Phase 3 research effort will focus on four key attributes: flexibility, interaction, the ability to take advantage of past experience, and compatibility with present and future Phase 4 goals. The proposed navigation task will provide an efficient transition to Phase 4, which requires a system demonstration using facilities at the Aviation Research Laboratory (ARL). The ARL, through ongoing research has developed physical resources and a capability for research on navigation problems. By orienting the CADM task toward this general area, we can take advantage of equipments and software that already exist. The flexibility is dictated by the multi-mission multi-profile aircraft that will operate during the 1980's. Interaction is necessary for pilot confidence and for efficiency in dynamic task allocation. Ability to change parameters, procedures, or structure based on prior failures and successes seems basic to intelligence, whether real or artificial. Our thrust will be to provide an expanded airplane subsystem, an advanced CADM capable of operating in environments containing some uncertainty, and an expanded and improved pilot-CADM interface.