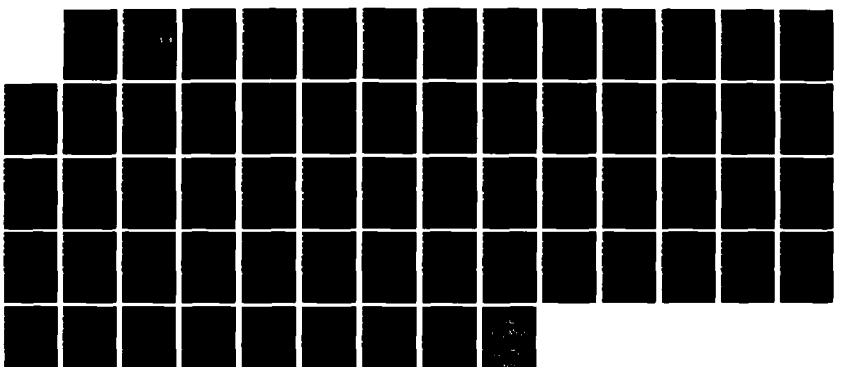
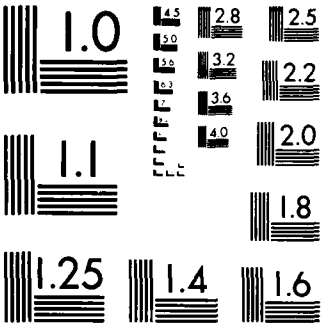
AD-A165 145 A LISP-BASED EXPERT SYSTEM FOR DETECTING FAILURES IN AIRCRAFT SYSTEMS (U) PRINCETON UNIV NJ DEPT OF MECHANICAL AND AEROSPACE ENGINERRIN C Y HUANG 1/1

UNCLASSIFIED 30 JUN 84 MAE-1666 ARO-20155.3-MA F/G 6/4 NL





Princeton University

Department of
Mechanical and
Aerospace Engineering

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A 165 145

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. N/A	3. RECIPIENT'S CATALOG NUMBER N/A
4. TITLE (and Subtitle) A LISP-BASED EXPERT SYSTEM FOR DETECTING FAILURES IN AIRCRAFT SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Interim Technical Report Jan. 1/84 - June 30/84
		6. PERFORMING ORG. REPORT NUMBER 1666-MAE
7. AUTHOR(s) Chien Huang		8. CONTRACT OR GRANT NUMBER(s) DAAG29-84-K-0048
9. PERFORMING ORGANIZATION NAME AND ADDRESS Princeton University MAE Department Princeton, NJ 08544		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS N/A
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office Post Office Box 12211 Research Triangle Park, NC 27709		12. REPORT DATE June 30, 1984
		13. NUMBER OF PAGES 55
14. MONITORING AGEN NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial Intelligence; Control Systems; Flight Control; Cybernetics; Computer Systems; Systems Analysis

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An expert system that provides fault detection and limited fault tolerance was designed for a generic aircraft model. A "best-first" search among applicable rules was utilized, leading to efficient recognition and processing of abnormal situations. Results showed that such an expert system is feasible and that it could contribute to increased reliability of aircraft system operation without a high degree of complexity.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

1666-MAE

A LISP-BASED EXPERT SYSTEM FOR DETECTING FAILURES IN AIRCRAFT SYSTEMS

Chien Y. Huang
June 30, 1984

Interim Technical Report
U.S. ARMY RESEARCH OFFICE
Contract No. DAAG29-84-K-0048

PRINCETON UNIVERSITY
Department of Mechanical and Aerospace Engineering
Princeton, NJ 08544

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

ABSTRACT

An expert system that provides fault detection and limited fault tolerance was designed for a generic aircraft model. A "best-first" search among applicable rules was utilized, leading to efficient recognition and processing of abnormal situations. Results showed that such an expert system is feasible and that it could contribute to increased reliability of aircraft system operation without a high degree of complexity.

TABLE OF CONTENTS

Abstract ii

page

1. INTRODUCTION 1
2. SYSTEM BASELINE 3
3. SYSTEM DATABASES 6
4. THE RULE SPACE 10
5. THE EXPERT SYSTEM (FDIES) 13
6. A SAMPLE RUN OF THE EXPERT SYSTEM 17
7. CONCLUSIONS AND FUTURE WORK 22

BIBLIOGRAPHY 23

Appendix page

A. LIST OF DEVICES 24
B. THE BULLETIN DATABASE 26
C. THE TIGER.DATA DATABASE 28
D. PROGRAM LISTING OF THE RULES 47
E. THE FDIES SOURCE CODE 49

LIST OF TABLES

Table		page
1.	The Rules	12
2.	Some Utility Procedures of FDIES	16

LIST OF FIGURES

Figure		page
1.	Fault Detection and Identification Expert System	3
2.	The Generic Aircraft	4
3.	An Example of Output from BULLETIN	6
4.	An Example of Adding an Entry to TIGER.DATA	7
5.	Typical Entry of an Inter-equiv Table	8
6.	Searching in Rule Space	11
7.	Flow Chart of FDIES	14
8.	Case I: Faulty Tank	19
9.	Case II: Differing Readings	20
10.	Case III: Abnormal Engine	21

1. INTRODUCTION

Modern aircraft systems are increasingly sophisticated in response to demands for higher performance. This trend has resulted in the design of complicated control modules whose continuous service is the key to a safe and successful flight. The general strategy used to ensure online operation of these units is to provide enough physical and analytical redundancy so that individual component failures do not jeopardize a mission. Although it is crucial for the systems to have backups, it is equally important to supply means to detect when they have suffered a degradation or even loss of performance. Many sensors are added to accomplish these supervisory tasks, but in doing so we have only accentuated the need for more fault tolerance and detection.

An analysis of the situation shows that there is an information overload, which calls for an intelligent decision-maker that can oversee the status of each system and take appropriate actions based on the incoming data. This requirement can be satisfactorily met by utilizing some of the results from the emerging field of artificial intelligence (AI).

Artificial Intelligence is often associated with natural language processing and mathematical theorem proving, but it has expanded in recent years to include other areas, such as automated decision making using expert systems. An expert system attempts to pool the knowledge of experts into production rules, which are usually in the form of "if-and-then" statements. Its purpose is to provide the users with a tool to solve relatively complicated problems by comparing different courses of action an ' choosing what appears to be the best.

The alternative that AI offers to the previous problem, therefore, is to design a system which is sufficiently smart to figure out if a malfunction has occurred and which is capable of taking remedial actions. The long-term objectives of the present work are to develop and configure necessary hardware and software for a system that will tackle the issues just raised. The ultimate goal is to test the functionality and robustness of such a system onboard airplanes or helicopters in a realistic scenario. The short-term objectives are to ascertain the feasibility of the AI approach as well as its complexity.

In the following text we describe the preliminary implementation of an expert system, called FDIES (Fault Detection and Identification Expert System), which addresses some of the issues of fault-detection. A block diagram of it is shown in Fig. 1 . Section 2 outlines the baseline for the generic aircraft used in our study. Section 3 describes the system databases, which are the knowledge source for the expert system. Section 4 explains the rules, which constitute the reasoning power of FDIES. Section 5 details the operation of FDIES. Section 6 gives a sample run of FDIES for three cases. Finally Section 7 contains the conclusions and statements of the future work.

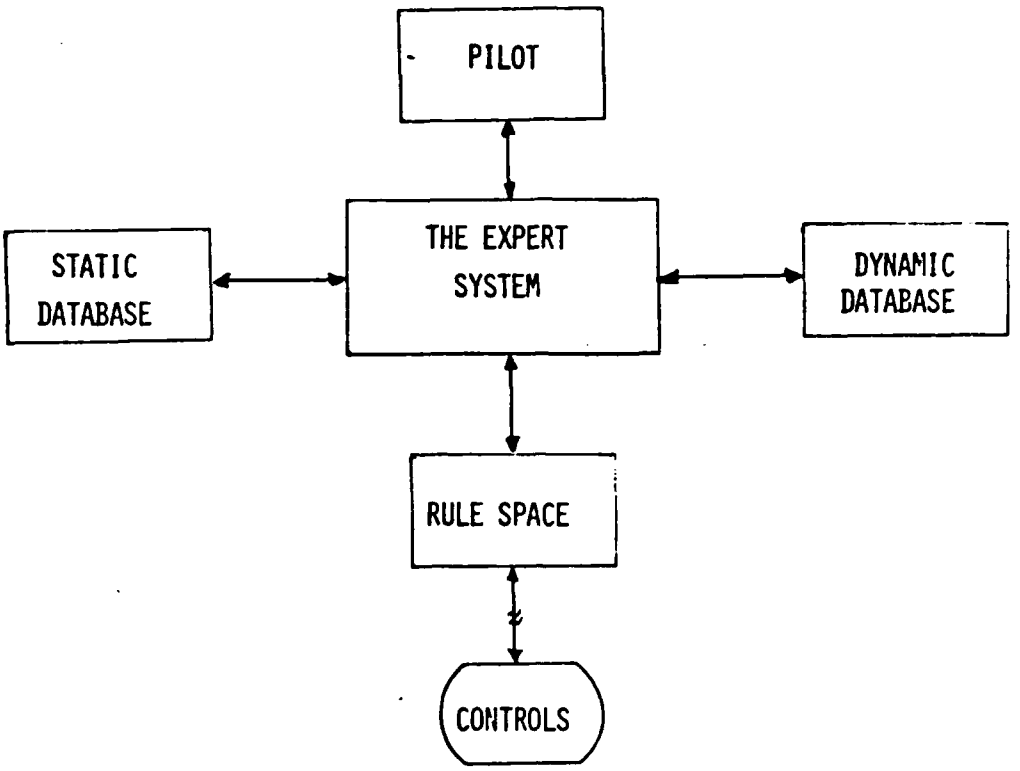


Figure 1: Fault Detection and Identification Expert System Block Diagram

2. SYSTEM BASELINE

We chose to deal with a generic airplane model which has a set of components that normally would be found in a modern aircraft. These components are located in five sections of the aircraft: left wing, right wing, nose, tail, and cockpit (Fig. 2). A sixth section, rotor, is available when we are dealing with helicopters. A list of specific devices assumed to be on the aircraft is found in Appendix A.

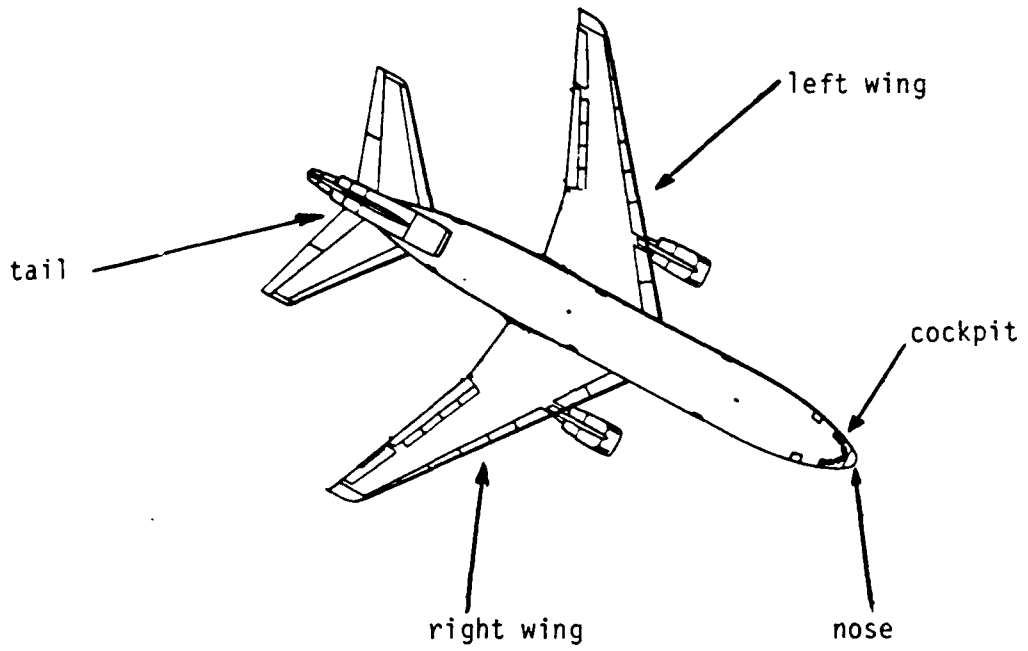


Figure 2: The Generic Aircraft

Several assumptions were made: an airspeed-sensor plus a backup are at the tip of the left-wing and at the nose. Static pressure sensors are at the roots of the each wing. Engines are mounted on the wings as well in the tail section. Ailerons are split into two independent sections, one at tip and other at the mid-section of the wing. All control surfaces are driven by doubly redundant electric actuators; hydraulic actuators would be handled in similar fashion. All sensor outputs are converted to digital format and checked by the onboard computer before being displayed on CRTs in the cockpit. Many devices are physically redundant, and a number of them have analytical equivalents as well. The devices generally are linked or related to other devices. These

and other pertinent data are all available (as property lists) to the expert system in a database called TIGER.DATA. The current readings and status of devices are posted in a separate database called BULLETIN. These two databases are described in more detail in the section that follows.

The programming language employed is LISP, which is widely used by the AI community. Its advantages are ease of constructing and modifying the databases (in form of lists), as well as good symbolic manipulation capability. The disadvantage is that running LISP may require long processing time; with the current rate of increase in computation speeds, this should soon be a minor issue.

3. SYSTEM DATABASES

The expert system obtains information concerning the aircraft system's current structure and status from two databases: one dynamic (BULLETIN), and the other static (TIGER.DATA).

The dynamic database, BULLETIN (Appendix B), contains the readings and status of all devices; it is constantly updated by a separate front-end system which will be part of our implementation at a later stage. BULLETIN also keeps a list of abnormal devices, that is, any devices that are either faulty or "down." Any alert on the abnormal behaviour of a device is also posted here. This is similar to the "blackboard" approach used in the HEARSAY system [1]. The format of BULLETIN is shown in Fig. 3 . The section name is listed first, followed by readings or status of all devices within that section. The last entries under the heading of "abnormal-devices" are reserved for any new alert and for keeping track of all faulty devices.

```
(left-wing
  ((airspeed-sensor self 50. phys-equiv 50. analy-equiv 50.)
   (engine ok)...)
 right-wing
  ((engine down)
   (tank ok)...)
 ...
 ...
 abnormal-devices
  ((left-wing aileron-1 new-alert)
   (right-wing engine down)))
```

Figure 3: An Example of Output from BULLETIN

The static database, TIGER.DATA, possesses a complete record of sections' devices (see Appendix C). It is actually implemented by attaching facts concerning the device as the property-list of the section's name under the device's name. This is fashioned after Minsky's frame approach [2]. The database is built by calling the procedure DEVICE, as illustrated in the next figure.

```
(device actuator-1
  IN left-wing
  ((status ok)
   (power-source batt-1-node)
   (function actuator)
   (location middle)
   (same-location (engine temp-sensor rpm-sensor ...))
   (related-devices (actuator-2))
   (gain 10.0)
   (linked-to ((stick-1 stick-2) (aileron-1 aileron-2)))
   (physical-equiv actuator-2)))
```

Figure 4: An Example of Adding an Entry to TIGER.DATA

After the evaluation of the procedure DEVICE shown in Fig. 4, FDIES will record the device "actuator-1" under the left-wing in its database. Its default assumption is that the left-wing actuator-1's status is "ok"; that its power source is "battery-1"; that it is located at the middle of the left wing; that at the same location we find also "engine", "temp-sensor",...; that it is linked forward to "stick-1" and "stick-2" and backward to "aileron-1" and "aileron-2." In the event that a device (actuator-1) is lost, the devices of the first list may still be able to perform their functions if they have other links (actuator-2), but the devices on the second-list are uncontrollable by the lost device (but are certainly controllable by its physical equivalent). Thus, the devices in

the first list will be marked as "broken-link", whereas those on the second list will be marked "unusable."

On many occasions a device in one section is connected to devices of other sections. In our previous example, the left-wing's "actuator-1" is connected to the cockpit's "stick-1" and "stick-2." Since it is not correct to declare devices for one section in other sections, yet it is crucial for the expert system to know how devices are linked, we include a special entry, "inter-equiv", under each section. It plays the role of an interconnection map among different sections of the aircraft. This is illustrated in Fig. 5 .

```
(device inter-equiv
  OF left-wing
  ((batt-1 (nose batt-1))
   (batt-2 (nose batt-2))
   ...
   (stick-1 (cockpit stick-1))
```

Figure 5: Typical Entry of an Inter-equiv Table

When searching through the database, if FDIES encounters any component that is not explicitly declared within a section, it automatically looks in the "inter-equiv" table to see if it is declared in another section. The necessity of inter-equiv becomes apparent when we consider the situation in which the expert system discovers that battery-1 (which is resident in the nose section) is drained. It then must find all devices (many of which are in other sections) and mark them "unusable". FDIES accomplishes this task by looking up battery-1's "link-to" list and the inter-equiv table.

In summary, FDIES's knowledge is contained in BULLETIN and TIGER.DATA. By accessing them, it remains informed of the relationships among devices and of each device's status.

4. THE RULE SPACE

The expert system currently utilizes ten rules ,which are divided into four groups, to handle failures. Group 1 consists of Rules #1, #2, and #10. They are used to mark devices in the databases as "down" or faulty or to restore them to normal status, respectively. They also are responsible for propagating the effects of the fault to the devices which are linked to the failed one. Group 2 has Rules #3 and #4; they are invoked when a device's reading differs from its equivalent device. Group 3 consists of Rule #9 alone, whose application occurs when the pilot makes an inquiry regarding, for instance, an ill-behaving device; in this case FDIES simply responds to the pilot's commands. Finally, Group 4 encompasses Rules #5 to Rule #8; these rules are evoked to decipher any alert related to abnormal components. A complete program listing of these rules can be found in Appendix D. A verbal description of each rule is presented in Table 1 .

Note that we have implicitly guided our application of rules . When a device is found abnormal, then Rule #5 is first checked to see if all devices in the same location are also abnormal. If not, then Rule #6 is called. It Rule #6 is not applicable, it will try Rule #7. Rule#8 will apply only when either Rule #5 or Rule #6 is satisfied. See Fig. 6 .

The "if" parts of these rules are not compared exhaustively against the current alerts. FDIES retrieves and applies only those rules which are pertinent to the situation. Usually one or two will suffice. This is a simple form of "best-first" search, which is the technique that focuses the attention of an expert system to the most promising node [3].

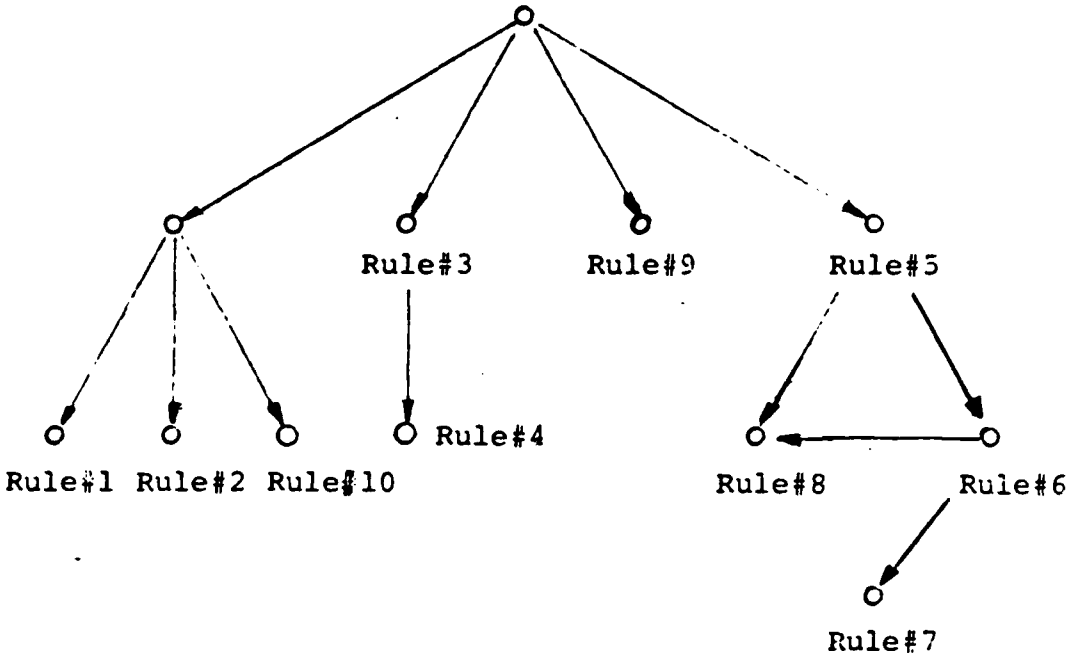


Figure 6: Searching in Rule Space

TABLE 1

The Rules

RULE#1:

IF a device is alerted as down,
THEN mark it "down". This also has the side effect of marking the "linked-to" devices as "unusable" or "broken-link."

RULE#2:

IF a device is alerted as faulty,
THEN mark it faulty.

RULE#3:

IF a device's reading is different from that of its equivalent,
THEN check both against the third equivalent's reading (if it exists). If two agree, then the third one is declared faulty (by issuing an alert on the BULLETIN). If none are the same, then the rule fails.

RULE#4:

IF a device's behavior is different from its analytical equivalent,
THEN see if a new gain or bias can be found by applying the routine "generate-and-test."

RULE#5:

IF a device's reading is considered abnormal,
THEN check to see if other devices in the common location are abnormal as well. If so, apply Rule #8; otherwise apply Rule #6.

RULE#6:

IF a device's reading is considered abnormal, and it has been determined that some devices in the same location are normal,
THEN check to see if related devices are abnormal. If so, apply Rule #8; otherwise apply Rule #7.

TABLE 1 (concluded)

RULE#7:

IF a device's reading is abnormal, and some related devices have been determined to be normal,
THEN the device is faulty; issue an alert to this effect. (Implicitly, this has the side-effect of removing any action caused by the abnormality; e.g., an engine shutdown due to false reading of a temperature sensor).

RULE#8:

IF devices common to a particular characteristic are all found to be abnormal,
THEN declare all of them to be faulty and report the nature of the problem.

RULE#9:

IF the alert is a request by the pilot to check a device's value,
THEN compare what the expert system knows to what the pilot believes. This command is usually confined to display devices that are in the cockpit.

RULE#10:

IF the alert says that a device has returned to normal,
THEN restore the status of the device to "ok" (including its "linked-to" devices).

5. THE EXPERT SYSTEM (FDIES)

The expert system's primary task is to assure normal operation of the aircraft. At present, it can only decide when an alert may be false, when a device is faulty, or when parts of aircraft are damaged. Ultimately, it will be able to produce new control strategies in the face of abnormalities enabling the flight to continue in safety.

FDIES is made up of several key procedure calls (Appendix E). A flow chart of the system is given in Fig. 7 . The procedure

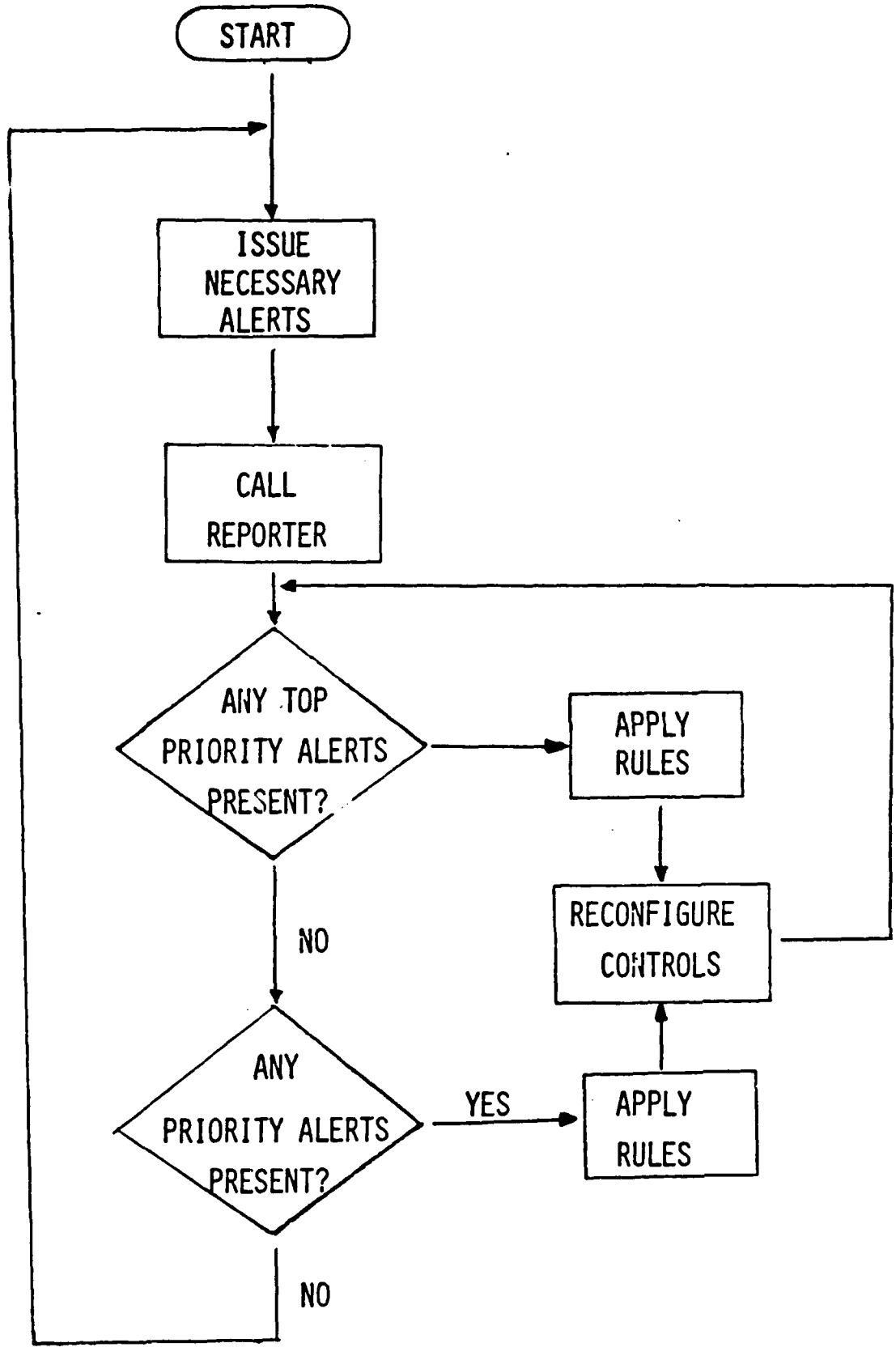


Figure 7: Flow Chart of FDIES

REPORTER checks the BULLETIN database for irregularities and warnings. If any are found, REPORTER issues an alert of the proper type (which is placed on either the high-priority stack or the priority stack). FDIES then "pops" the top alert off the stack and applies the rules which belong to that particular alert (by using the procedure APPLY-RULES). FDIES iterates to make sure that all alerts (which are often issued within the rule being applied) are processed.

Other procedures of interest which are not directly utilized by FDIES are listed in Table 2 . Some play important roles in connection with successful application of rules (e.g. the procedure SIMILAR), and others provide an interactive interface between the expert system and the pilot (e.g., the procedure CHANGE-BULLETIN).

TABLE 2

Some Utility Procedures of FDIES

ISSUE-ALERT: Places alert given on the stack.

MARK-DEVICE: Marks the given device with proper value; i.e., it changes the status of the device (and its "linked-to" counterparts) on BULLETIN and TIGER.DATA.

CHANGE: Changes the value of a specified slot within a section's device.

CHANGE-BULLETIN: Changes an entry in BULLETIN.

RETRIEVE: Returns the value of the slot of the given section device presently in the TIGER.DATA.

DEVICE-DESCRIPTION: Returns the entire fact field pertaining to a particular device.

SIMILAR: Matches two patterns. It also sets any variable beginning with "?" in one pattern to its corresponding value in the other pattern.

RESTORE: Restores the device's status to "ok."

GENERATE-AND-TEST: Finds new gain or bias by trial and error. Currently returns "true" for all calls.

6. A SAMPLE RUN OF THE EXPERT SYSTEM

The following pages illustrate some examples of FDIES's functionality. We have divided the output roughly into three columns. The first column shows the operator input to the system. The second one, in which each entry is preceded by "&&", contains the response from FDIES. The third column displays comments which are in capital letters and are preceded by "***."

Three different kinds of failure and their ramifications are analyzed. The first one is the simple case of a device (left wing tank) being found faulty (Fig. 8). In this situation, FDIES put the tank on the abnormal device list and declared its "link-to" devices (engine, level-sensor) as "unusable." The second one (Fig. 9) dealt with a discrepancy in readings of two equivalent devices (airspeed-sensor-1 and -2). A check with their analytical equivalent showed that sensor-2 was faulty, so FDIES issued an alert to put airspeed-sensor-2 on the abnormal device list. Finally the third one (Fig. 10) was concerned with a device (left wing engine) being declared abnormal. Since there was no supporting evidence that anything else was wrong, FDIES placed the engine on the abnormal device list.

```
Script started on Tue May 22 09:09:11 1984
%lisp                                                          *** ENTER LISP

-> (dskin fdies tiger.rules tiger.data temp-bulletin)          *** DISK IN ALL NECESSARY FILES. NOTE THAT A MUCH SMALLER
                                                               *** DATABASE, TEMP-BULLETIN, IS USED IN PLACED OF
                                                               *** THE FULL-BLOWN VERSION

                &&[load fdies]
                &&[load tiger.rules]
                &&[load tiger.data]
                &&[load temp-bulletin]
                &&(fdies tiger.rules tiger.data temp-bulletin)
                                                               *** CHECK WHAT WE HAVE IN BULLETIN
-> bulletin
                &&(left-wing ((airspeed-sensor-1 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
                              (airspeed-sensor-2 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
                              (level-sensor-1 self 1000.0) (engine self ok) (tank self ok)
                              (temp-sensor self 500.0) (rpm-sensor self 10000.0)
                              (pressure-sensor 100.0)) abnormal-devices (nil))
                                                               *** EVERYTHING IS FINE. NO ABNORMAL CONDITION PRESENT

(fdies)                                                        *** CALL FDIES

                &&(no (more) alerts present)                   *** OBVIOUSLY NOTHING IS HAPPENING...
                &&(to quit.. type q. Otherwise hit return)
                &&*(you are in input-mode. To run FDIES type "p")

                                                               *** NOW NOTIFY THE FDIES THAT LEFT-WING'S TANK IS DOWN

(issue-alert '(down (left-wing tank]
                &&((issue-alert '(down (left-wing tank))) is: ((down (left-wing tank))))
                                                               *** AND RUN THE SYSTEM!
p
                &&(no (more) alerts present)                   *** THE ALERT HAS BEEN TAKEN CARE OF.
                &&(to quit.. type q. Otherwise hit return)
                &&*(you are in input-mode. To run FDIES type "p")
                                                               *** NOW CHECK THE BULLETIN TO SEE THE CHANGE
bulletin                                                       *** AS EXPECTED NOT ONLY TANK WAS DECLARED FAULTY
                                                               *** AND ADDED TO THE ABNORMAL DEVICE LIST,
                                                               *** BUT ALSO ENGINE AND LEVEL-SENSOR-1 WERE DECLARED UNUSABL
                &&(bulletin is: (left-wing ((airspeed-sensor-1 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
                                            (airspeed-sensor-2 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
                                            (level-sensor-1 self unusable) (engine self unusable)
                                            (tank self down)
                                            (temp-sensor self 500.0) (rpm-sensor self 10000.0)
                                            (pressure-sensor 100.0))
                                abnormal-devices (nil (left-wing tank down))))
```

Figure 8: Case I: Faulty Tank

```
(device-description left-wing tank)                                    *** LET'S CHECK THE TIGER.DATA AS WELL
                &&((device-description left-wing tank) is:             *** FIRST WE CHECK THE LEFT-WING'S TANK
                        ((status down)                                 *** AND WE SEE THAT IT IS DOWN
                         (function storage) (location root)
                         (same-location (static-pressure-sensor-1 static-pressure-sensor-2
                                         level-sensor-1 batt-1-node batt-2-node))
                         (related-devices (level-sensor-1))
                         (linked-to (nil (engine level-sensor-1)))))
(device-description left-wing en***eTHENTHE LEFT-WING'S ENGINE
                &&((device-description left-wing engine) is:
                        ((status unusable)                             *** WHICH IS UNUSABLE SINCE TANK IS DOWN
                         (power-source batt-1-node)
                         (function propulsion) (location middle)
                         (same-location (temp-sensor rpm-sensor
                                         pressure-sensor aileron-2 actuator-1 actuator-2))
                         (linked-to ((tank) (temp-sensor rpm-sensor pressure-sensor)))))

(issue-alert '(normal (left-wing tank]                                 *** NOW INFORM THE TIGER THAT TANK IS FIXED.
                &&(((issue-alert '(normal (left-wing tank))) is:
                   ((normal (left-wing tank))))
p                                                                      *** RUN THE SYSTEM AGAIN.

                &&(no (more) alerts present)                           *** AGAIN THE FLYING TIGER TOOK CARE OF THE ALERT.
                &&(to quit.. type q. Otherwise hit return)
                &&*(you are in input-mode. To run FDIES type "p")
bulletin                                                               *** WE CHECK THE BULLETIN AND FIND THAT
                                                                       *** FDIES HAS RETURNED TANK,
                                                                       *** ENGINE AND LEVEL-SENSOR-1
                                                                       *** TO THEIR NEW NORMAL STATUS.
                &&(bulletin is:
                  (left-wing ((airspeed-sensor-1 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
                              (airspeed-sensor-2 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
                              (level-sensor-1 self ok) (engine self ok) (tank self ok)
                              (temp-sensor self 500.0) (rpm-sensor self 10000.0)
                              (pressure-sensor 100.0))
                    abnormal-devices (nil nil)))                       *** NO DEVICE IS ABNORMAL...

(device-description left-wing tank)                                    *** WE CHECK ALSO THE TIGER.DATA
                                                                       *** FOR THE STATUS OF LEFT-WING TANK
                &&((device-description left-wing tank) is:
                   ((status ok) (function storage) (location root)
                    (same-location (static-pressure-sensor-1 static-pressure-sensor-2
                                    level-sensor-1 batt-1-node batt-2-node))
                    (related-devices (level-sensor-1))
                    (linked-to (nil (engine level-sensor-1))))))
```

Figure 8 (concluded)

```
                                                        *** NOW A NEW TYPE OF FAILURE, WE FORCE
                                                        *** THE LEFT-WING AIRSPEED-SENSOR-1'S
                                                        *** READING TO BE DIFFERENT FROM ITS PHYSICAL EQUIVALENT

(change-bulletin left-wing airspeed-sensor-1 phys-equiv 5Ø.Ø)
                &&((change-bulletin left-wing airspeed-sensor-1 phys-equiv 5Ø.Ø) is:
                   (5Ø.Ø analy-equiv 1ØØ.Ø))

bulletin                                                *** WE CHECK THE BULLETIN
                &&(bulletin is:                         *** NOTE THAT CHANGE IS THE FIRST ENTRY
                  (left-wing ((airspeed-sensor-1 self 1ØØ.Ø phys-equiv 5Ø.Ø analy-equiv 1ØØ.Ø)
                              (airspeed-sensor-2 self 1ØØ.Ø phys-equiv 1ØØ.Ø analy-equiv 1ØØ.Ø)
                              (level-sensor-1 self ok) (engine self ok) (tank self ok)
                              (temp-sensor self 5ØØ.Ø) (rpm-sensor self 1ØØØØ.Ø)
                              (pressure-sensor 1ØØ.Ø))
                   abnormal-devices (nil nil)))
p                                                       *** RUN!

(rule#4 does not apply)                                 *** RULE#4 OBVIOUSLY DOES NOT APPLY

                &&(no (more) alerts present)            *** DONE
                &&(to quit.. type q. Otherwise hit return)
                &&"(you are in input-mode. To run FDIES type "p")

bulletin                                                *** A CHECK OF BULLETIN SHOWS THAT
                                                        *** LEFT-WING AIRSPEED-SENSOR-2 HAS BEEN
                                                        *** DECLARED FAULTY AS EXPECTED SINCE IT IS THE
                                                        *** SENSOR-1'S PHYSICAL EQUIVALENT.
                &&(bulletin is:
                  (left-wing ((airspeed-sensor-1 self 1ØØ.Ø phys-equiv 5Ø.Ø analy-equiv 1ØØ.Ø)
                              (airspeed-sensor-2 self faulty phys-equiv 1ØØ.Ø analy-equiv 1ØØ.Ø)
                              (level-sensor-1 self ok) (engine self ok) (tank self ok)
                              (temp-sensor self 5ØØ.Ø) (rpm-sensor self 1ØØØØ.Ø)
                              (pressure-sensor 1ØØ.Ø))
                   abnormal-devices (nil nil (left-wing airspeed-sensor-2 faulty))))
                                                        *** AND IT HAS ALSO BEEN ADDED TO THE
                                                        *** ABNORMAL DEVICE LIST.

(device-description left-wing airspeed-sensor-2)        *** TIGER.DATA HAS ALSO BEEN CHANGED...
                &&((device-description left-wing airspeed-sensor-2) is:
                        ((status faulty) (power-source batt-2-node)
                         (function sensor) (location tip)
                         (same-location (airspeed-sensor-1 aileron-1))
                         (related-devices (airspeed-sensor-1))
                         (linked-to ((A/D2)))
                         (physical-equiv airspeed-sensor-1)
                         (analytical-equiv if-needed-sensor)))

                                                        *** WE RETURN THE SITUATION TO NORMAL

(change-bulletin left-wing airspeed-sensor-1 phys-equiv 1ØØ.Ø)
                &&((change-bulletin left-wing airspeed-sensor-1 phys-equiv 1ØØ.Ø) is:
                   (1ØØ.Ø analy-equiv 1ØØ.Ø))
```

Figure 9: Case II: Differing Readings

```
                                                                    *** LAST EXAMPLE.
                                                                    *** A DEVICE HAS JUST BEEN DECLARED ABNORMAL
                                                                    *** NOTE HOW FDIES WILL ADD THE DEVICE
                                                                    *** TO THE ABNORMAL-LIST AND SET OTHER
                                                                    *** RELEVANT ENTRIES TO UNUSABLE OR BROKEN-LINK
(add-abnormal-devices left-wing engine new-alert)                   *** LEFT-WING ENGINE HAS BEEN ALERTED TO
                                                                    *** BE ABNORMAL
                &&((add-abnormal-devices left-wing engine new-alert) is:
                   (nil nil (left-wing airspeed-sensor-2 faulty)
                            (left-wing engine new-alert)))

(issue-alert '(normal (left-wing airspeed-sensor-2)                 *** SIMULTANEOUSLY WE RESTORE THE STATUS OF
                                                                    *** AIRSPEED-SENSOR-2 TO NORMAL.
                                                                    *** GO!
P
                &&(no (more) alerts present)                        *** DONE AGAIN.
                &&(to quit type q. Otherwise hit return)
                &&"(you are in input-mode. To run FLLYING-TIGER type "p")

bulletin                                                            *** WE CHECK THE BULLETIN. WE NOTICE THAT
                                                                    *** AIRSPEED-SENSOR-2 IS OK NOW AND
                                                                    *** ENGINE IS DECLARED FAULTY
                &&(left-wing ((airspeed-sensor-1 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
                              (airspeed-sensor-2 self ok phys-equiv 100.0 analy-equiv 100.0)
                              (level-sensor-1 self ok) (engine self faulty)
                              (tank self broken-link)               *** AND LINK FROM ENGINE TO TANK IS BROKEN;
                              (temp-sensor self unusable)           *** AND BOTH TEMPERATURE SENSOR
                              (rpm-sensor self unusable)            *** AND RPM SENSOR ARE UNSUSABLE
                              (pressure-sensor 100.0))
                 abnormal-devices (nil nil nil (left-wing engine faulty)))

                                                                    *** THAT'S ALL FOLKS ***
script done on Tue May 22 09:35:11 1984
```

Figure 10: Case III: Abnormal Engine

7. CONCLUSIONS AND FUTURE WORK

An expert system that performs fault detection and displays some of the features of fault tolerance has been developed. The former is exemplified by the detection and identification of the discrepancy of equivalent sensors; the latter is illustrated by the choice of the healthy sensor in face of the discrepancy, thus avoiding the use of misleading data.

In light of these results, we believe that our work has demonstrated the feasibility of a fault-detection expert system, that it assesses the degree of difficulty and complexity, and that it provides a prototype on which future systems can be based.

Looking ahead, there is a great deal of work yet to be done. Notable is the addition of the function that will be able to reconfigure the aircraft's control system so as to minimize the effects of the fault. A prime candidate for consideration is the multiple-model-estimation approach. Another important objective is to imbed in the system the capability to write new rules automatically when unknown situations are encountered. This belongs to the area of machine learning. Finally, we also will study teaching methodologies that will enable experts to augment our knowledge base on fault detection. Implementation of these facilities will bring us closer to truly fault-tolerant systems.

BIBLIOGRAPHY

1. Erman L.D. and Lesser V.R. "The HEARSAY-II Speech Understanding System: A Tutorial," In **Trends in Speech Recognition**, W.Lea (Ed), Prentice-Hall, Englewood Cliffs, NJ, 1980, pp. 361-381.

2. Minsky, M. "A Framework for Representing Knowledge." In **The Psychology of Computer Vision**, P.Winston (Ed), McGraw-Hill Co., New York, 1975, pp. 211-277.

3. Rich, E. **Artificial Intelligence**, McGraw-Hill Inc., New York, 1983, pp. 78-86.

Appendix A

LIST OF DEVICES

This appendix contains a list of devices that the expert system currently recognizes.

LEFT-WING: airspeed-sensor-1 airspeed-sensor-2 static-pressure-sensor-1 static-pressure-sensor-2 aileron-1 aileron-2 actuator-1 actuator-2 tank level-sensor-1 engine temp-sensor rpm-sensor pressure-sensor batt-1-node batt-2-node.

RIGHT-WING: static-pressure-sensor-1 static-pressure-sensor-2 aileron-1 aileron-2 actuator-1 actuator-2 tank level-sensor-1 engine temp-sensor rpm-sensor pressure-sensor batt-1-node batt-2-node.

NOSE: airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1 gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1 accelerometer-2 radar-1 antenna.

TAIL: engine temp-sensor rpm-sensor pressure-sensor rudder actuator-1 actuator-2 elevator-1 elevator-2 actuator-3 actuator-4 batt-1-node batt-2-node. COCKPIT: stick-1 stick-2 pedal-1 pedal-2 radio-1 airspeed-ind-1 airspeed-ind-2 climb-rate-ind-1 climb-rate-ind-2 engines-rpm-ind-1 engines-rpm-ind-2 manifold-pressure-ind-1 manifold-pressure-ind-2 engines-temp-ind-1 engines-temp-ind-2 atmos-

pheric-pressure-ind-1 atmospheric-pressure-ind-2 fuel-gauge-1 fuel-gauge-2 horizon-ind-1 horizon-ind-2 altitude-ind-1 altitude-ind-2 display-driver-1 display-driver-2 A/D1-8 A/D9-16 A/D17-24 A/D25-32 batt-1-node batt-2-node.

ROTOR: main-rotor transmission engine tank level-sensor-1 temp-sensor rpm-sensor pressure-sensor swashplate batt-1-node.

Appendix B

THE BULLETIN DATABASE

This appendix contains the complete listing of the dynamic database currently used by FDIES.

```
~
(setq bulletin
  '(left-wing
    ((airspeed-sensor-1 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
     (airspeed-sensor-2 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
     (static-sensor-1 self 12.0 phys-equiv 12.0 analy-equiv 12.5)
     (static-sensor-2 self 12.0 phys-equiv 12.0 analy-equiv 12.5)
     (level-sensor-1 self 1000.0)
     (temp-sensor self 500.0)
     (rpm-sensor self 10000.0)
     (pressure-sensor 100.0)
     (aileron-1 self ok)
     (aileron-2 self ok)
     (actuator-1 self ok)
     (actuator-2 self ok)
     (tank self ok)
     (engine 0k)
     (batt-1-node self 24.0)
     (batt-2-node self 24.0))
    right-wing
    ((static-sensor-1 self 12.0 phys-equiv 12.0 analy-equiv 12.5)
     (static-sensor-2 self 12.0 phys-equiv 12.0 analy-equiv 12.5)
     (level-sensor-1 self 1000.0)
     (temp-sensor self 500.0)
     (rpm-sensor self 10000.0)
     (pressure-sensor 100.0)
     (aileron-1 self ok)
     (aileron-2 self ok)
     (actuator-1 self ok)
     (actuator-2 self ok)
     (tank self ok)
     (engine self ok)
     (batt-1-node self 24.0)
     (batt-2-node self 24.0))
    nose
    ((airspeed-sensor-1 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
     (airspeed-sensor-2 self 100.0 phys-equiv 100.0 analy-equiv 100.0)
     (batt-1 self 24.0)
     (batt-2 self 24.0)
     (gyro-1.1 self 1.0)
     (gyro-1.2 self 2.1)
     (gyro-1.3 self 0.5)
     (gyro-2.1 self 1.0)
     (gyro-2.2 self 2.1)
     (gyro 2.3 self 0.5)
     (antenna self ok)
     (radar-1 self ok)
     (accelerometer-1 self 5.0)
     (accelerometer-2 self 2.0))
    tail
    ((temp-sensor self 500.0)
     (rpm-sensor self 10000.0)
     (pressure-sensor self 100.0)
     (rudder self ok)
     (elevator-1 self ok)
```

```
 (elevator-2 self ok)
 (actuator-1 self ok)
 (actuator-2 self ok)
 (actuator-3 self ok)
 (actuator-4 self ok)
 (batt-1-node self 24.0)
 (batt-2-node self 24.0))
cockpit
((airspeed-ind-1 self 100.0 phys-equiv 100.0)
 (airspeed-ind-2 self 100.0 phys-equiv 100.0)
 (climb-rate-ind-1 self 0.5 phys-equiv 0.5)
 (climb-rate-ind-2 self 0.5 phys-equiv 0.5)
 (engines-rpm-ind-1 self (10000.0 10000.0 10000.0) phys-equiv (10000.0 10000.0 10000.0))
 (engines-rpm-ind-2 self (10000.0 10000.0 10000.0) phys-equiv (10000.0 10000.0 10000.0))
 (manifold-pressure-ind-1 self (100.0 100.0 100.0) phys-equiv (100.0 100.0 100.0))
 (manifold-pressure-ind-2 self (100.0 100.0 100.0) phys-equiv (100.0 100.0 100.0))
 (engines-temp-ind-1 self (500.0 500.0 500.0) phys-equiv (500.0 500.0 500.0))
 (engines-temp-ind-2 self (500.0 500.0 500.0) phys-equiv (500.0 500.0 500.0))
 (atmospheric-pressure-ind-1 self 11.5 phys-equiv 11.5)
 (atmospheric-pressure-ind-2 self 11.5 phys-equiv 11.5)
 (fuel-gauge-1 self 1000.0 phys-equiv 1000.0)
 (fuel-gauge-2 self 1000.0 phys-equiv 1000.0)
 (altitude-ind-1 self 1000.0 phys-equiv 1000.0)
 (altitude-ind-2 self 1000.0 phys-equiv 1000.0)
 (stick-1 self ok)
 (stick-2 self ok)
 (pedal-1 self ok)
 (pedal-2 self ok)
 (radio-1 self ok)
 (horizon-ind-1 self ok)
 (horizon-ind-2 self ok)
 (display-driver-1 self ok)
 (display-driver-2 self ok)
 (A/D1-8 self ok)
 (A/D9-16 self ok)
 (A/D17-24 self ok)
 (A/D15-32 self ok)
 (batt-1-node self 24.0)
 (batt-2-node self 24.0))
rotor
((level-sensor-1 self 1000.0)
 (temp-sensor self 500.0)
 (rpm-sensor self 10000.0)
 (pressure-sensor self 100.0)
 (engine self ok)
 (swashplate self ok)
 (main-rotor self ok)
 (batt-1-node self 24.0))
abnormal-devices
~example (section-name device-name status (down, faulty or new-alert))
(nil)))
~
~
```

Appendix C

THE TIGER.DATA DATABASE

This appendix contains the static database declarations which are made before the expert system is run.

```
~this file contains the data for the FDIES
~
~first make known all the sections of the airplane
~
(sections left-wing right-wing tail nose cockpit rotor)
~
~*************** LEFT-WING **************
~
(device airspeed-sensor-1
   IN left-wing
   ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (location tip)
    (same-location (airspeed-sensor-2 aileron-1))
    (related-devices (airspeed-sensor-2))
    (linked-to ((A/D1)))
    (physical-equiv airspeed-sensor-2)
    (analytical-equiv if-needed-sensor)))
~
(device airspeed-sensor-2
   IN left-wing
   ((status ok)
    (power-source batt-2-node)
    (function sensor)
    (location tip)
    (same-location (airspeed-sensor-1 aileron-1))
    (related-devices (airspeed-sensor-1))
    (linked-to ((A/D2)))
    (physical-equiv airspeed-sensor-1)
    (analytical-equiv if-needed-sensor)))
~
(device static-pressure-sensor-1
   IN left-wing
   ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (location root)
    (same-location (static-pressure-sensor-2 batt-1-node batt-2-node tank level-sensor-1))
    (related-devices (static-pressure-sensor-2))
    (linked-to ((A/D3)))
    (physical-equiv static-pressure-sensor-2)
    (analytical-equiv if-needed-sensor2)))
~
(device static-pressure-sensor-2
   IN left-wing
   ((status ok)
    (power-source batt-2-node)
    (function sensor)
    (location root)
    (same-location (static-pressure-sensor-1 batt-1-node batt-2-node tank level-sensor-1))
    (related-devices (static-pressure-sensor-1))
    (linked-to ((A/D4)))
    (physical-equiv static-pressure-sensor-1)
    (analytical-equiv if-needed-sensor2)))
```

```
~
(device aileron-1
   IN left-wing
   ((status ok)
    (function effector)
    (location tip)
    (same-location (airspeed-sensor-1 airspeed-sensor-2))
    (related-devices (actuator-1 actuator-2))
    (linked-to ((actuator-1 actuator-2)))))
~
(device aileron-2
   IN left-wing
   ((status ok)
    (funtion effector)
    (location middle)
    (same-location (engine temp-sensor rpm-sensor pressure-sensor actuator-1
                    actuator-2))
    (related-devices (actuator-1 actuator-2))
    (linked-to ((actuator-2)))))
~
(device actuator-1
   IN left-wing
   ((status ok)
    (power-source batt-1-node)
    (function actuator)
    (location middle)
    (same-location (engine temp-sensor rpm-sensor pressure-sensor aileron-2
                    actuator-2))
    (related-devices (actuator-2))
    (gain 10.0)
    (linked-to ((stick-1 stick-2) (aileron-1 aileron-2)))
    (physical-equiv actuator-2)))
~
(device actuator-2
   IN left-wing
   ((status ok)
    (power-source batt-2-node)
    (function actuator)
    (location middle)
    (same-location (engine temp-sensor rpm-sensor pressure-sensor aileron-2
                    actuator-1))
    (related-devices (actuator-1))
    (gain 10.0)
    (linked-to ((stick-1 stick-2) (aileron)))
    (physical-equiv actuator-1)))
~
(device tank
   IN left-wing
   ((status ok)
    (function storage)
    (location root)
    (same-location (static-pressure-sensor-1 static-pressure-sensor-2
                    level-sensor-1 batt-1-node batt-2-node))
    (related-devices (level-sensor-1))
    (linked-to (() (engine level-sensor-1)))))
~
(device level-sensor-1
   IN left-wing
   ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (location root)
    (same-location (static-pressure-sensor-1 static-pressure-sensor-2 tank
                    batt-1-node batt-2-node))
    (linked-to ((tank A/D5)))))
```

```
~
(device engine
  IN left-wing
  ((status ok)
   (power-source batt-1-node)
   (function propulsion)
   (location middle)
   (same-location (temp-sensor rpm-sensor pressure-sensor aileron-2 actuator-1
                   actuator-2))
   (linked-to ((tank) (temp-sensor rpm-sensor pressure-sensor)))))
~
(device temp-sensor
  IN left-wing
  ((status ok)
   (power-source batt-1-node)
   (function sensor)
   (location middle)
   (same-location (engine rpm-sensor pressure-sensor aileron-2 actuator-1 actuator-2))
   (related-devices (rmp-sensor pressure-sensor))
   (linked-to ((engine A/D6)))))
~
(device rpm-sensor
  IN left-wing
  ((status ok)
   (power-source batt-1-node)
   (function sensor)
   (location middle)
   (same-location (engine temp-sensor pressure-sensor aileron-2 actuator-1 actuator-2))
   (related-devices (temp-sensor pressure-sensor))
   (linked-to ((engine A/D7)))))
~
(device pressure-sensor
  IN left-wing
  ((status ok)
   (power-source batt-1-node)
   (function sensor)
   (location middle)
   (same-location (engine rpm-sensor pressure-sensor aileron-2 actuator-1 actuator-2))
   (related-devices (temp-sensor rpm-sensor))
   (linked-to ((engine A/D8)))))
~
(device batt-1-node
  IN left-wing
  ((status ok)
   (function battery)
   (location root)
   (same-location (static-pressure-sensor-1 static-pressure-sensor-2 tank
                   level-sensor-1))
   (linked-to ((batt-1) (airspeed-sensor-1 static-pressure-sensor-1 actuator-1
                engine level-sensor-1 temp-sensor rpm-sensor pressure-sensor)))))
~
(device batt-2-node
  IN left-wing
  ((status ok)
   (function battery)
   (same-location (static-pressure-sensor-1 static-pressure-sensor-1 tank
                   level-sensor-1))
   (linked-to ((batt-2) (airspeed-sensor-2 static-pressure-sensor-2 actuator-2)))))
~
(device inter-equiv
  OF left-wing
  ((batt-1 (nose batt-1))
   (batt-2 (nose batt-2))
   (A/D1 (cockpit A/D1-8))
   (A/D2 (cockpit A/D25-32))
```

```
(A/D3 (cockpit A/D1-8))
(A/D4 (cockpit A/D25-32))
(A/D5 (cockpit A/D1-8))
(A/D6 (cockpit A/D1-8))
(A/D7 (cockpit A/D1-8))
(A/D8 (cockpit A/D1-8))
(stick-1 (cockpit stick-1))
(stick-2 (cockpit stick-2))))
~
```

```
~•••••••••••••••• RIGHT-WING ••••••••••••••
~
(device static-pressure-sensor-1
  IN right-wing
  ((status ok)
   (power-source batt-1-node)
   (function sensor)
   (location root)
   (same-location (static-pressure-sensor-2 batt-1-node batt-2-node tank level-sensor-1))
   (related-devices (static-pressure-sensor-2))
   (linked-to ((A/D3)))
   (physical-equiv static-pressure-sensor-2)
   (analytical-equiv if-needed-sensor2)))
~
(device static-pressure-sensor-2
  IN right-wing
  ((status ok)
   (power-source batt-2-node)
   (function sensor)
   (location root)
   (same-location (static-pressure-sensor-1 batt-1-node batt-2-node tank level-sensor-1))
   (related-devices (static-pressure-sensor-1))
   (linked-to ((A/D4)))
   (physical-equiv static-pressure-sensor-1)
   (analytical-equiv if-needed-sensor2)))
~
(device aileron-1
  IN right-wing
  ((status ok)
   (function effector)
   (location tip)
   (same-location (airspeed-sensor-2))
   (related-devices (actuator-1 actuator-2))
   (linked-to ((actuator-1 actuator-2)))))
~
(device aileron-2
  IN right-wing
  ((status ok)
   (funtion effector)
   (location middle)
   (same-location (engine temp-sensor rpm-sensor pressure-sensor actuator-1
                   actuator-2))
   (related-devices (actuator-1 actuator-2))
   (linked-to ((actuator-2)))))
~
(device actuator-1
  IN right-wing
  ((status ok)
   (power-source batt-1-node)
   (function actuator)
   (location middle)
   (same-location (engine temp-sensor rpm-sensor pressure-sensor aileron-2
                   actuator-2))
   (related-devices (actuator-2))
   (gain 10.0)
   (linked-to ((stick-1 stick-2) (aileron-1 aileron-2)))
   (physical-equiv actuator-2)))
~
(device actuator-2
  IN right-wing
  ((status ok)
   (power-source batt-2-node)
   (function actuator)
   (location middle)
```

```
    (same-location (engine temp-sensor rpm-sensor pressure-sensor aileron-2
                    actuator-1))
    (related-devices (actuator-1))
    (gain 10.0)
    (linked-to ((stick-1 stick-2) (aileron)))
    (physical-equiv actuator-1)))
~
(device tank
  IN right-wing
  ((status ok)
    (function storage)
    (location root)
    (same-location (static-pressure-sensor-1 static-pressure-sensor-2
                    level-sensor-1 batt-1-node batt-2-node))
    (related-devices (level-sensor-1))
    (linked-to (() (engine level-sensor-1)))))
~
(device level-sensor-1
  IN right-wing
  ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (location root)
    (same-location (static-pressure-sensor-1 static-pressure-sensor-2 tank
                    batt-1-node batt-2-node))
    (linked-to ((tank A/D5)))))
~
(device engine
  IN right-wing
  ((status ok)
    (power-source batt-1-node)
    (function propulsion)
    (location middle)
    (same-location (temp-sensor rpm-sensor pressure-sensor aileron-2 actuator-1
                    actuator-2))
    (linked-to ((tank) (temp-sensor rpm-sensor pressure-sensor)))))
~
(device temp-sensor
  IN right-wing
  ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (location middle)
    (same-location (engine rpm-sensor pressure-sensor aileron-2 actuator-1 actuator-2))
    (related-devices (rmp-sensor pressure-sensor))
    (linked-to ((engine A/D6)))))
~
(device rpm-sensor
  IN right-wing
  ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (location middle)
    (same-location (engine temp-sensor pressure-sensor aileron-2 actuator-1 actuator-2))
    (related-devices (temp-sensor pressure-sensor))
    (linked-to ((engine A/D7)))))
~
(device pressure-sensor
  IN right-wing
  ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (location middle)
    (same-location (engine rpm-sensor pressure-sensor aileron-2 actuator-1 actuator-2))
    (related-devices (temp-sensor rpm-sensor))
```

```
   (linked-to ((engine A/D8)))))
~
(device batt-1-node
  IN right-wing
  ((status ok)
   (function battery)
   (location root)
   (same-location (static-pressure-sensor-1 static-pressure-sensor-2 tank
                   level-sensor-1))
   (linked-to ((batt-1) (static-pressure-sensor-1 actuator-1
                         engine level-sensor-1 temp-sensor rpm-sensor pressure-sensor))
~
(device batt-2-node
  IN right-wing
  ((status ok)
   (function battery)
   (same-location (static-pressure-sensor-1 static-pressure-sensor-1 tank
                   level-sensor-1))
   (linked-to ((batt-2) (airspeed-sensor-2 static-pressure-sensor-2 actuator-2)))))
~
(device inter-equiv
  OF right-wing
  ((batt-1 (nose batt-1))
   (batt-2 (nose batt-2))
   (A/D3 (cockpit A/D1-8))
   (A/D4 (cockpit A/D25-32))
   (A/D5 (cockpit A/D1-8))
   (A/D6 (cockpit A/D9-16))
   (A/D7 (cockpit A/D9-16))
   (A/D8 (cockpit A/D9-16))
   (stick-1 (cockpit stick-1))
   (stick-2 (cockpit stick-2))))
```

```
~ ********** NOSE SECTION **********
~
(device airspeed-sensor-1
  IN nose
  ((status ok)
   (power-source batt-1)
   (function sensor)
   (same-location (airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                   gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                   accelerometer-2 radar-1 antenna))
   (related-devices (airspeed-sensor-1))
   (linked-to ((A/D1)))
   (physical-equiv airpseed-sensor-2)
   (analytical-equiv if-needed-sensor1)))
~
(device airspeed-sensor-2
  IN nose
  ((status ok)
   (power-source batt-2)
   (function sensor)
   (same-location (airspeed-sensor-1 batt-1 batt-2 gyro-1.1
                   gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                   accelerometer-2 radar-1 antenna))
   (related-devices (airspeed-sensor-1))
   (linked-to ((A/D2)))
   (physical-equiv airspeed-sensor-1)
   (analytical-equiv if-needed-sensor1)))
~
(device batt-1
  IN nose
  ((status ok)
   (function battery)
   (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-2 gyro-1.1
                   gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                   accelerometer-2 radar-1 antenna))
   (linked-to (() (batt-1-lwing batt-1-rwing batt-1-tail batt-1-cockpit
                   batt-1-rotor airspeed-sensor-1 gyro-1 accelerometer radar-1)))))
~
(device batt-2
  IN nose
  ((status ok)
   (function battery)
   (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 gyro-1.1
                   gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                   accelerometer-2 radar-1 antenna))
   (linked-to (() (batt-2-lwing batt-2-rwing batt-2-tail batt-2-cockpit
                   airspeed-sensor-2 gyro-2)))))
~
(device gyro-1.1
  IN nose
  ((status ok)
   (power-source batt-1)
   (function sensor)
   (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2
                   gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                   accelerometer-2 radar-1 antenna))
   (related-devices (gyro-1.2 gyro-1.3))
   (linked-to ((A/D3)))
   (physical-equiv gyro-2.1)))
~
(device gyro-1.2
  IN nose
  ((status ok)
   (power-source batt-1)
```

```
    (function sensor)
    (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                    gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                    accelerometer-2 radar-1 antenna))
    (related-devices (gyro-1.1 gyro-1.3))
    (linked-to ((A/D4)))
    (physical-equiv gyro-2.2)))
~
(device gyro-1.3
  IN nose
  ((status ok)
    (power-source batt-1)
    (function sensor)
    (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                    gyro-1.2 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                    accelerometer-2 radar-1 antenna))
    (related-devices (gyro-1.1 gyro-1.2))
    (linked-to ((A/D5)))
    (physical-equiv gyro-2.3)))
~
(device gyro-2.1
  IN nose
  ((status ok)
    (power-source batt-2)
    (function sensor)
    (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                    gyro-1.2 gyro-1.3 gyro-2.2 gyro-2.3 accelerometer-1
                    accelerometer-2 radar-1 antenna))
    (related-devices (gyro-2.2 gyro-2.3))
    (linked-to ((A/D6)))
    (physical-equiv gyro-1.1)))
~
(device gyro-2.2
  IN nose
  ((status ok)
    (power-source batt-2)
    (function sensor)
    (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                    gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.3 accelerometer-1
                    accelerometer-2 radar-1 antenna))
    (related-devices (gyro-2.1 gyro-2.3))
    (linked-to ((A/D7)))
    (physical-equiv gyro-1.2)))
~
(device gyro-2.3
  IN nose
  ((status ok)
    (power-source batt-2)
    (function sensor)
    (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                    gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 accelerometer-1
                    accelerometer-2 radar-1 antenna))
    (related-devices (gyro-2.1 gyro-2.2))
    (linked-to ((A/D8)))
    (physical-equiv gyro-1.3)))
~
(device accelerometer-1
  IN nose
  ((status ok)
    (power-source batt-1)
    (function sensor)
    (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                    gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3
                    accelerometer-2 radar-1 antenna))
    (related-devices (accelerometer-2))
```

```
    (linked-to ((A/D9)))))
~
(device accelerometer-2
   IN nose
   ((status ok)'
    (power-source batt-2)
    (function sensor)
    (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                    gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                    radar-1 antenna))
    (related-devices (accelerometer-1))
    (linked-to ((A/D10)))))
~
(device radar-1
   IN nose
   ((status ok)
    (power-source batt-1)
    (function sensor)
    (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                    gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                    accelerometer-2 antenna))
    (linked-to ((parallel-input)))))
~
(device antenna
   IN nose
   ((status ok)
    (function communication)
    (same-location (airspeed-sensor-1 airspeed-sensor-2 batt-1 batt-2 gyro-1.1
                    gyro-1.2 gyro-1.3 gyro-2.1 gyro-2.2 gyro-2.3 accelerometer-1
                    accelerometer-2 radar-1))
    (linked-to ((radio)))))

(device inter-equiv
   OF nose
   ((batt-1-lwing (tail batt-1-node))
    (batt-1-rwing (right-wing batt-1-node))
    (batt-1-tail (tail batt-1-node))
    (batt-1-cockpit (cockpit batt-1-node))
    (batt-1-rotor (rotor batt-1-node))
    (batt-2-lwing (tail batt-2-node))
    (batt-2-rwing (right-wing batt-2-node))
    (batt-2-tail (tail batt-2-node))
    (batt-2-cockpit (cockpit batt-2-node))
    (A/D1 (cockpit A/D9-16))
    (A/D2 (cockpit A/D9-16))
    (A/D3 (cockpit A/D9-16))
    (A/D4 (cockpit A/D9-16))
    (A/D5 (cockpit A/D9-16))
    (A/D6 (cockpit A/D25-32))
    (A/D7 (cockpit A/D25-32))
    (A/D8 (cockpit A/D25-32))
    (A/D9 (cockpit A/D17-24))
    (A/D10 (cockpit A/D17-24))
    (radio (cockpit radio))
    (paralell-input (cockpit parallel-input))))

~•••••••••••••• TAIL SECTION ••••••••••••••••
~
(device engine
   IN tail
   ((status ok)
    (power-source batt-1-node)
    (function propulsion)
    (same-location (temp-sensor rpm-sensor pressure-sensor rudder actuator-1
                    actuator-2 actuator-3 actuator-4))
```

```
    (related-devices (temp-sensor rpm-sensor pressure-sensor))
    (linked-to ((tank) (temp-sensor rpm-sensor pressure-sensor)))))
~
(device temp-sensor
  IN tail
  ((status ok)
   (power-source batt-1-node)
   (function sensor)
   (same-location (engine rpm-sensor pressure-sensor rudder actuator-1
                    actuator-2 actuator-3 actuator-4))
   (related-devices (rpm-sensor pressure-sensor))
   (linked-to ((engine A/D1)))))
~
(device rpm-sensor
  IN tail
  ((status ok)
   (power-source batt-1-node)
   (function sensor)
   (same-location (engine temp-sensor pressure-sensor rudder actuator-1
                    actuator-2 actuator-3 actuator-4))
   (related-devices (temp-sensor pressure-sensor))
   (linked-to ((engine A/D2)))))
~
(device pressure-sensor
  IN tail
  ((status ok)
   (power-source batt-1-node)
   (function sensor)
   (same-location (engine temp-sensor rpm-sensor rudder actuator-1
                    actuator-2 actuator-3 actuator-4))
   (related-devices (temp-sensor rpm-sensor))
   (linked-to ((engine A/D3)))))
~
(device rudder
  IN tail
  ((status ok)
   (function effector)
   (same-location (engine temp-sensor rpm-sensor pressure-sensor actuator-1
                    actuator-2 actuator-3 actuator-4))
   (linked-to ((actuator-1 actuator-2)))))
~
(device actuator-1
  IN tail
  ((status ok)
   (power-source batt-1-node)
   (function actuator)
   (gain 10.0)
   (same-location (engine temp-sensor rpm-sensor pressure-sensor rudder
                    actuator-2 actuator-3 actuator-4))
   (linked-to ((pedal-1 pedal-2) (rudder)))
   (physical-equiv actuator-2)))
~
(device actuator-2
  IN tail
  ((status ok)
   (power-source batt-2-node)
   (function actuator)
   (gain 10.0)
   (same-location (engine temp-sensor rpm-sensor pressure-sensor rudder
                    actuator-1 actuator-3 actuator-4))
   (linked-to ((pedal-1 pedal-2) (rudder)))
   (physical-equiv actuator-1)))
~
(device elevator-1
  IN tail
```

```
  ((status ok)
   (location right)
   (function effector)
   (linked-to ((actuator-2)))))
~
(device elevator-2
  IN tail
  ((status ok)
   (location left)
   (function effector)
   (linked-to ((actuator-2)))))
~
(device actuator-3
  IN tail
  ((status ok)
   (power-source batt-1-node)
   (function actuator)
   (gain 10.0)
   (same-location (engine temp-sensor rpm-sensor pressure-sensor rudder
                  actuator-1 actuator-2 actuator-4))
   (linked-to ((stick-1 stick-2) (elevator-1 elevator-2)))
   (physical-equiv actuator-4))))
~
(device actuator-4
  IN tail
  ((status ok)
   (power-source batt-2-node)
   (function actuator)
   (gain 10.0)
   (same-location (engine temp-sensor rpm-sensor pressure-sensor rudder
                  actuator-1 actuator-2 actuator-3))
   (linked-to ((stick-1 stick-2) (elevator-1 elevator-2)))
   (physical-equiv actuator-3))))
~
(device batt-1-node
  IN tail
  ((status ok)
   (function battery)
   (linked-to ((batt-1) (actuator-1 actuator-3 engine temp-sensor rpm-sensor
                        pressure-sensor)))))
~
(device batt-2-node
  IN tail
  ((status ok)
   (function battery)
   (linked-to ((batt-2) (actuator-2 actuator-4)))))
~
(device inter-equiv
  OF tail
  ((batt-1-node (nose batt-1))
   (batt-2-node (nose batt-2))
   (A/D1 (cockpit A/D17-24))
   (A/D2 (cockpit A/D17-24))
   (A/D3 (cockpit A/D17-24))
   (stick-1 (cockpit stick-1))
   (stick-2 (cockpit stick-2))
   (pedal-1 (cockpit pedal-1))
   (pedal-2 (cockpit pedal-2))))
~
```

```
~ ••••••••••••••••• COCKPIT SECTION ••••••••••••••••••
~
(device stick-1
  IN cockpit
  ((status ok)
   (function control)
   (linked-to (() (actuator-1 actuator-2 actuator-3 actuator-4)))
   (physical-equiv stick-2)))
~
(device stick-2
  IN cockpit
  ((status ok)
   (function control)
   (linked-to (() (actuator-1 actuator-2 actuator-3 actuator-4)))
   (physical-equiv stick-1)))
~
(device pedal-1
  IN cockpit
  ((status ok)
   (function control)
   (linked-to (() (actuator-5 actuator-6)))
   (physical-equiv pedal-2)))
~
(device pedal-2
  IN cockpit
  ((status ok)
   (function control)
   (linked-to (() (actuator-5 actuator-6)))
   (physical-equiv pedal-1)))
~
(device radio-1
  IN cockpit
  ((status ok)
   (power-source batt-1-node)
   (function communication)
   (linked-to (() (antenna)))))
~
(device airspeed-ind-1
  IN cockpit
  ((status ok)
   (power-source batt-1-node)
   (function display)
   (linked-to ((display-driver)))
   (physical-equiv airspeed-ind-2)))
~
(device airspeed-ind-2
  IN cockpit
  ((status ok)
   (power-source batt-2-node)
   (function display)
   (linked-to ((display-driver-2)))
   (physical-equiv airspeed-ind-1)))
~
(device climb-rate-ind-1
  IN cockpit
  ((status ok)
   (power-source batt-1-node)
   (function display)
   (linked-to ((display-driver-1)))
   (physical-equiv climb-rate-ind-2)))
~
(device climb-rate-ind-2
  IN cockpit
  ((status ok)
   (power-source batt-2-node)
```

```
    (linked-to ((display-driver-2)))
    (physical-equiv atmospheric-pressure-ind-1)))
~
(device fuel-gauge-1
   IN cockpit
   ((status ok)
    (power-source batt-1-node)
    (function display)
    (linked-to ((display-driver-1)))
    (physical-equiv fuel-gauge-2)))
~
(device fuel-gauge-2
   IN cockpit
   ((status ok)
    (power-source batt-2-node)
    (function display)
    (linked-to ((display-driver-2)))
    (physical-equiv fuel-gauge-1)))
~
(device horizon-ind-1
   IN cockpit
   ((status ok)
    (power-source batt-1-node)
    (function display)
    (linked-to ((display-driver-1)))
    (physical-equiv horizon-ind-2)))
~
(device horizon-ind-1
   IN cockpit
   ((status ok)
    (power-source batt-2-node)
    (function display)
    (linked-to ((display-driver-2)))
    (physical-equiv horizon-ind-1)))
~
(device altitude-ind-1
   IN cockpit
   ((status ok)
    (power-source batt-1-node)
    (function display)
    (linked-to ((display-driver-1)))
    (physical-equiv altitude-ind-2)))
~
(device altitude-ind-2
   IN cockpit
   ((status ok)
    (power-source batt-2-node)
    (function display)
    (linked-to ((display-driver-2)))
    (physical-equiv altitude-ind-1)))
~
(device display-driver-1
   IN cockpit
   ((status ok)
    (power-source batt-1-node)
    (function driver)
    (linked-to ((****a-lot of devices ****)))
    (physical-equiv display-driver-2)))
~
(device display-driver-2
   IN cockpit
   ((status ok)
    (power-source batt-2-node)
    (function driver)
    (linked-to ((*** a lot of devices ***)))
```

```
    (function display)
    (linked-to ((display-driver-2)))
    (physical-equiv climb-rate-ind-1)))
~
(device engines-rpm-ind-1
   IN cockpit
   ((status ok)
    (power-source batt-1-node)
    (function display)
    (linked-to ((display-driver-1)))
    (physical-equiv engines-rpm-ind-2)))
~
(device engines-rpm-ind-2
   IN cockpit
   ((status ok)
    (power-source batt-2-node)
    (function display)
    (linked-to ((display-driver-2)))
    (physical-equiv engines-rpm-ind-1)))
~
(device manifold-pressure-ind-1
   IN cockpit
   ((status ok)
    (power-source batt-1-node)
    (function display)
    (linked-to ((display-driver-1)))
    (physical-equiv manifold-pressure-ind-2)))
~
(device manifold-pressure-ind-2
   IN cockpit
   ((status ok)
    (power-source batt-2-node)
    (function display)
    (linked-to ((display-driver-2)))
    (physical-equiv manifold-pressure-ind-1)))
~
(device engines-temp-ind-1
   IN cockpit
   ((status ok)
    (power-source batt-1-node)
    (function display)
    (linked-to ((display-driver-1)))
    (physical-equiv engines-temp-ind-2)))
~
(device engines-temp-ind-2
   IN cockpit
   ((status ok)
    (power-source batt-2-node)
    (function display)
    (linked-to ((display-driver-2)))
    (physical-equiv engines-temp-ind-1)))
~
(device atmospheric-pressure-ind-1
   IN cockpit
   ((status ok)
    (power-source batt-1-node)
    (function display)
    (linked-to ((display-driver-1)))
    (physical-equiv atmospheric-pressure-ind-2)))
~
(device atmospheric-pressure-ind-2
   IN cockpit
   ((status ok)
    (power-source batt-2-node)
    (function display)
```

```
   (physical-equiv display-driver-1)))
~
(device A/D1-8
  IN cockpit
  ((status ok)
   (power-source batt-1-node)
   (function converter)
   (linked-to ((sensor-1 sensor-2 sensor-3 sensor-4 sensor-5 sensor-6 sensor-7
                sensor-8)))))
~
(device A/D9-16
  IN cockpit
  ((status ok)
   (power source batt-1-node)
   (function converter)
   (linked-to ((sensor-9 sensor-10 sensor-11 sensor-12 sensor-13 sensor-14
                sensor-15 sensor-16)))))
~
(device A/D17-24
  IN cockpit
  ((status ok)
   (power-source batt-1-node)
   (function converter)
   (linked-to ((sensor-17 sensor-18 sensor-19 sensor-20 sensor-21 sensor-22
                sensor-23 sensor-24)))))
~
(device A/D25-32
  IN cockpit
  ((status ok)
   (power-source batt-2-node)
   (function converter)
   (linked-to ((sensor-25 sensor-26 sensor-27 sensor-28 sensor-29 sensor-30
                sensor-31)))))
~
(device batt-1-node
  IN cockpit
  ((status ok)
   (function battery)
   (linked-to ((batt-1) (radio-1 airspeed-ind-1 climb-rate-ind-1 fuel-gauge-1
                         engines-rpm-ind-1 manifold-pressure-ind-1
                         engines-temp-ind-1 horizon-ind-1 altitude-ind-1
                         display-driver-1 A/D1-8 A/D9-16 A/D17-24)))))
~
(device batt-2-node
  IN cockpit
  ((status ok)
   (linked-to ((batt-2) (airspeed-ind-2 climb-rate-ind-2 fuel-gauge-2
                         engines-rpm-ind-2 manifold-pressure-ind-2
                         engines-temp-ind-2 horizon-ind-2 altitude-ind-2
                         display-driver-2 A/D25-32)))))
~
(device inter-equiv
  OF cockpit
  ((batt-1 (nose batt-1))
   (batt-2 (nose batt-2))
   (actuator-1 (left-wing actuator))
   (actuator-2 (right-wing actuator))
   (actuator-3 (tail actuator-1))
   (actuator-4 (tail actuator-2))
   (actuator-5 (tail actuator-3))
   (actuator-6 (tail-actuator-4))
   (sensor-1 (left-wing airpseed-sensor-1))
   (sensor-2 (left-wing static-pressure-sensor-1))
   (sensor-3 (left-wing level-sensor-1))
   (sensor-4 (left-wing temp-sensor))
```

```
(sensor-5 (left-wing rpm-sensor))
(sensor-6 (left-wing pressure-sensor))
(sensor-7 (right-wing static-sensor-1))
(sensor-8 (right-wing level-sensor-1))
(sensor-9 (right-wing temp-sensor))
(sensor-10 (right-wing rpm-sensor))
(sensor-11 (right-wing pressure-sensor))
(sensor-12 (nose airspeed-sensor-1))
(sensor-13 (nose gyro-1))
(sensor-14 (nose gyro-1))
(sensor-15 (nose gyro-1))
(sensor-16 (nose accelerometer))
(sensor-17 (nose accelerometer))
(sensor-18 (tail temp-sensor))
(sensor-19 (tail rpm-sensor))
(sensor-20 (tail pressure-sensor))
(sensor-21 (rotor level-sensor-1))
(sensor-22 (rotor temp-sensor))
(sensor-23 (rotor rpm-sensor))
(sensor-24 (rotor pressure-sensor))
(sensor-25 (left-wing airspeed-sensor-2))
(sensor-26 (left-wing static-pressure-sensor-2))
(sensor-27 (right-wing static-pressure-sensor-2))
(sensor-28 (nose airspeed-pressure-sensor-2))
(sensor-29 (nose gyro-2))
(sensor-30 (nose gyro-2))
(sensor-31 (nose gyro-2))
(antenna (nose antenna))))
```

```
~ ••••••••••••••••••• MAIN-ROTOR (HELICOPTER ONLY) •••••••••••••••••••
~
(device main-rotor
   IN rotor
   ((status ok)
    (function propulsion)
    (linked-to ((transmission))))
~
(device transmission
   IN rotor
   ((status ok)
    (function driver)
    (linked-to ((engine) (main-rotor)))))
~
(device engine
   IN rotor
   ((status ok)
    (power-source batt-1-node)
    (function propulsion)
    (linked-to ((tank) (transmission temp-sensor rpm-sensor pressure-sensor)))))
~
(device tank
   IN rotor
   ((status ok)
    (function storage)
    (linked-to (() (engine level-sensor-1)))))
~
(device level-sensor-1
   IN rotor
   ((status ok)
    (power-source batt-node-1)
    (function sensor)
    (linked-to ((tank A/D5)))))
~
(device temp-sensor
   IN rotor
   ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (linked-to ((engine A/D6)))))
~
(device rpm-sensor
   IN rotor
   ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (linked-to ((engine A/D7)))))
~
(device pressure-sensor
   IN rotor
   ((status ok)
    (power-source batt-1-node)
    (function sensor)
    (linked-to ((engine A/D8)))))
~
(device swashplate
   IN rotor
   ((status ok)
    (function effector)
    (linked-to ((collective-stick)))))
~
(device batt-1-node
   IN rotor
   ((status ok)
    (function battery)
```

```
    (linked-to ((batt-1) (engine level-sensor-1 temp-sensor rpm-sensor
                                pressure-sensor)))))
~
(device inter-equiv
  OF rotor
  ((batt-1 (nose batt-1))
   (A/D5 (cockpit A/D17-24))
   (A/D6 (cockpit A/D17-24))
   (A/D7 (cockpit A/D17-24))
   (A/D8 (cockpit A/D33-40))
   (collective-stick (cockpit collective-stick-1)
                     (cockpit collective-stick-2))))
~
```

Appendix D

PROGRAM LISTING OF THE RULES

This appendix contains the source listing for the ten rules presently installed in the FDIES.

```
(rules rule#1
  IF      (down ?device)                                                    IF
  THEN    (mark-device (car ?device) (cadr ?device) 'down))                 THEN
~
(rules rule#2
  IF      (faulty ?device)                                                  IF
  THEN    (mark-device (car ?device) (cadr ?device) 'faulty))               THEN
~
(rules rule#3
  IF      (different ?device reading)                                       IF
  ~       from its physical equivalent is implicit here.
  THEN    (prog (temp-then section device)                                  THEN
                (setq section (car ?device))
                (setq device (cadr ?device))
                ~the device must have analytical equivalent
                (cond ((null (setq temp-then (look-up section device 'analy-equiv)))
                       (print '(no analytical equiv present... rule#3 failed)))
                      ~If it does then check to see to which reading it is equal.
                      (t (cond ((same (look-up section device 'self) temp-then)
                                (mark-device section
                                  (retrieve section device 'physical-equiv) 'faulty))
                               ((same (look-up section device 'phys-equiv) temp-then)
                                (mark-device section device 'faulty))
                               (t (print '(no readings are the same... rule#3 failed))))))))
~
(rules rule#4
  IF      (different ?device behavior)                                      IF
  THEN    (generate-and-test ?device))                                      THEN
~
(rules rule#5
  IF      (abnormal ?device reading)                                        IF
  THEN    (prog (other-devices device-list)                                 THEN
                ~first find the other devices in the same location
                (setq other-devices (retrieve (car ?device) (cadr ?device) 'same-location))
                ~If every device is on the abnormal list then apply rule#8
                (setq device-list other-devices)
                loop
                (cond ((null device-list)
                       (return (eval `(use-rule rule#8
                                                (common ,(cons ?device other-devices) location)))))
                      ((null (on-abnormal-list (list (car ?device) (car device-list))))
                       (return (eval `(use-rule rule#6
                                                (abnormal ,?device reading not-same-loc)))))
                      (t (setq device-list (cdr device-list))))
                (go loop)))
~
(rules rule#6
  IF      (abnormal ?device reading not-same-loc)                           IF
  THEN    (prog (other-devices device-list)                                 THEN
                (setq other-devices (retrieve (car ?device) (cadr ?device) 'related-devices))
                ~first check if there are related devices. Most of cases it will.
                ~If not apply rule#7
                (cond ((null other-devices)
                       (return (eval `(use-rule rule#7 (abnormal ,?device reading no-related)
                (setq device-list other-devices)
```

```
                                                                    ...THEN
                loop
                ~all devices on the list must be abnormal list to declare that
                ~all related devices faulty.
                (cond ((null device-list)
                       (return (eval `(use-rule rule#8
                                              (abnormal ,(cons ?device other-devices) related-dev
                      ((null (on-abnormal-list (list (car ?device) (car device-list))))
                       (return (eval `(use-rule rule#7 (abnormal ,?device reading no-related)
                      (t (setq device-list (cdr device-list))))
                (go loop)))
~
(rules rule#7
  IF      (abnormal ?device reading no-related)                                    IF
  ~if airplane's behavior is also abnormal, something serious is happening
  ~so many other rules will take care of the situation. Ignore the alert for now
  THEN  (cond ((on-abnormal-list 'airplane-behavior))                              THEN
              ~otherwise declare the device faulty.
              (t (issue-alert `(faulty ,?device)))))
~
(rules rule#8
  IF      (common ?device ?commonality)                                            IF
  THEN  (do ((device-list ?device (cdr device-list))                               THEN
             (a-device (car ?device)))
            ~after all devices have been declared faulty, report the
            ~nature of the problem.
            ((null device-list)
             (cond ((equal ?commonality 'location)
                    (setq temp (list (car a-device)
                                     (retrieve (car a-device) (cadr a-device) 'location)))
                    (print `(structural damage in ,(car a-device) ,temp)))
                   ((equal ?commonality 'related)
                    (print `(these related devices are down: ,?device)))))
            ~issue alert for each of the devices.
            (issue-alert `(faulty ,(car device-list)))))
~
(rules rule#9
  IF      (check ?device ?value)                                                   IF
  THEN  (prog (temp)                                                             THEN
              (setq temp (car ?device))
              ~first check if the device is in the cockpit
              (cond ((not (equal 'cockpit temp))
                     (return (print
                                '(check command must refer to a device in the cockpit))))
                    ~then check if the device is in bulletin
                    ((null (setq temp (look-up temp (cadr ?device) 'self)))
                     (return (print '(cannot check the device in question))))
                    ~finally check if the value is the same as reported.
                    ((same ?value temp)
                     (return (print `(,?device is OK))))
                    ~mark faulty if all fails
                    (t (issue-alert `(faulty ,?device))))))
~
(rules rule#10
  IF      (normal ?device)                                                         IF
  THEN  (restore ?device))                                                       THEN
~
~Put the rule number on the property list
(defprop down (rule#1) rule-number)
(defprop faulty (rule#2) rule-number)
(defprop different (rule#3 rule#4) rule-number)
(defprop abnormal (rule#5) rule-number)
(defprop common (rule#8) rule-number)
(defprop check (rule#9) rule-number)
(defprop normal (rule#10) rule-number)
~
```

Appendix E

THE FDIES SOURCE CODE

This appendix contains the complete program listing for the FDIES. It includes the main procedure FDIES as well as all the available utility procedures. Short comments on the purpose of each routine are given in the beginning of each major file.

```
~ This file contains executive procedure to run the FDIES
~ plus a few routines.
~
~fdies:              this is the top-level executive routine
~reporter:           checks the bulletin for abnormalities and issue proper alerts.
~my-gc:              removes nil's from the abnormal-devices list.
~auxiliary functions: used to retrieve data from the bulletin.
~issue-alert:        places the alert on either the high-priority or priority stack.
~push:               pushes an element on a stack.
~pop:                pops an element of a stack.
~apply-rules:        finds and applies correct rule to the alert in question.
~use-rule:           the routine that actually does the rule-application.
~
(defun fdies ()
  (prog (message top-alert)
        ~first some preliminaries
        (setq message '(you are in input-mode. To run FDIES type "p"))
        (setq high-p-stack nil)
        (setq p-stack nil)
        loop-1
        (terpr)
        (print '(no (more) alerts present)) (terpr)
        (print '(to quit.. type q. Otherwise hit return)) (terpr)
        ~Quit?
        (cond ((equal 'q (readc)) (return 'done)))
        ~otherwise sit in pause until the operator is ready
        (pause message)
        ~first check if there is any abnormalities
        (reporter)
        loop-2
        ~Any new alerts?
        (setq top-alert (pop 'high-priority))
        (cond ((null top-alert) (cond ((null (setq top-alert (pop 'priority)))
                                       (go loop-1)))))
        ~if yes apply the rules!
        (apply-rules top-alert)
        (go loop-2)))
~
(defun reporter ()
  (do ((news bulletin (cddr news))
       (section))
      ((null news))
      ~section name is usually the first item
      (setq section (car news))
      ~is it the abnormal-devices list? If yes take care of it.
```

```
        (cond ((equal section 'abnormal-devices)
               (do ((device-list (cadr news) (cdr device-list)))
                   ((null device-list))
                   ~is this item a new alert?
                   (cond ((equal 'new-alert (get-abnormal-status (car device-list)))
                          ~If yes, issue the alert. Otherwise do nothing
                          (setq section (get-abnormal-?device (car device-list)))
                          (issue-alert `(abnormal ,section reading))
                          ~Once alert is issued remove it from the abnormal list.
                          (rplaca device-list nil)))))
              ~if this is a section name, check for differences
              (t (do ((device-list (cadr news) (cdr device-list))
                      (phys-equiv-value) (self-value))
                     ((null device-list))
                     ~does a physical equivalent exist? If no do nothing
                     (cond ((null (setq phys-equiv-value (get-p-equiv (car device-list))))
                            ~are they numbers. If no do nothing as well
                        ((null (and (numbp (setq self-value (get-s-value (car device-list))))
                                    (numbp phys-equiv-value))))
                            ~Otherwise check if self-value and equiv value are same
                            ~If yes do nothing. If no issue alert.
                            (t (cond ((same self-value phys-equiv-value))
                                     (t
          (issue-alert `(different ,(list section
                                          (get-d-name (car device-list)))
                                   reading)))))))))))
~
(defun my-gc ()
  (do ((news bulletin (cddr news)))
      ((null news))
      (cond ((equal (car news) 'abnormal-devices)
             (return (rplaca (cdr news) (remove nil (cadr news))))))))
~
~Some auxiliary functions
~
(defun get-abnormal-status (a-list)
  (car (reverse a-list)))
~
(defun get-abnormal-?device (a-list)
  (reverse (cdr (reverse a-list))))
~
(defun get-p-equiv (a-list)
  (cadr (member 'phys-equiv a-list)))
~
(defun get-s-value (a-list)
  (caddr a-list))
~
(defun get-d-name (a-list)
  (car a-list))
~
(defun issue-alert (exp)
  (prog (temp)
        (setq temp (car exp))
        ~check to see if the alert is that of high-priority type
        (cond ((or (equal temp 'down) (equal temp 'faulty))
               ~if yes push the alert onto the high-priority stack
               (return (push exp 'high-priority)))
              ~otherwise push it on the priority stack
              (t (return (push exp 'priority))))))
~
(defun push (value stack-name)
  ~check to see which stack is to be used.
  (cond ((equal stack-name 'high-priority) (setq high-p-stack (cons value high-p-stack)))
        (t (setq p-stack (cons value p-stack)))))
~
```

```
(defun pop (stack-name)
  ~check which of the stack the value must be popped.
  (cond ((equal stack-name 'high-priority)
         (prog1 (car high-p-stack)
                (setq high-p-stack (cdr high-p-stack))))
        (t (prog1 (car p-stack)
                  (setq p-stack (cdr p-stack))))))
~
(defun apply-rules (alert)
  ~find out what rules are applicable
  (do ((starting-rules (get (car alert) 'rule-number) (cdr starting-rules))
       (?device) (?commonality) (?value))
      ~if no rules do nothing
      ((null starting-rules))
      ~apply use-rules iteratively until all rules are exhausted.
      (eval `(use-rule ,(car starting-rules) ,alert))))
~
(defmacro use-rule (rule# alert)
  ~first check if the rule is applicable
  (cond ((similar (get rule# 'if) alert)
         ~if so get the then part and evaluate it on the return (due to macro)
         (get rule# 'then))
        (t (print `(,rule# does not apply)))))
~ **********************************************************
~
~List of files:
~
~sections:          makes FDIES aware of sections being dealt.
~is-section:        checks to see if an element is part of section database already.
~mark-device:       marks device as well as those on the linked-to list to
~                   whatever status specified.
~                   or unusable.
~mark:              used by mark-devices to accomplish its goal.
~get-new-section:   gets the section that the device really belongs to.
~
(declare special frames)
(setq frames ())
~
(defun sections macro (exp)
  ~example (sections left-wing right-wing tail ....)
  (do ((temp (cdr exp) (cdr temp)))
      ((null temp) `',frames)
      ~check if the section is known already. If not add the the global frames
      (cond ((null (is-section (car temp))) (setq frames (cons (car temp) frames)))
            ~otherwise report existence and continue.
            (t (print `(,(car temp) is already known))))))
~
(defun is-section (x1)
  (cond ((member x1 frames) t)
        (t ())))
~
(defun mark-device (section device value)
  ~format is (mark-devices left-wing aileron down)
  (let ((temp ()) (value-1 'ok) (value-2 'ok))
       ~first check if the given section and device exist. If they don't
       ~do nothing. If they do set temp to the value of linked-to slot.
       (cond ((null (setq temp (is-present-1 section device 'linked-to))))
             ~Now iteratively change the status of the devices.
             ~first add the device on abnormal-devices list.
             (t (cond ((not (equal value 'ok))
                       (eval `(add-abnormal-devices ,section ,device ,value))
                       (setq value-1 'broken-link)
                       (setq value-2 'unusable)))
                ~then change its status on the bulletin and database.
                (eval `(change-bulletin ,section ,device self ,value))
```

```
                (eval `(change ,section ,device status to ,value))
                ~finally mark the devices on the first list of linked-to devices
                ~as broken-link
                (mark section (caar temp) value-1)
                ~and the devices on the second list as unusable.
                (mark section (cadar temp) value-2)))))
~
(defun mark (current-section device-list new-status)
  (do ((temp1 device-list (cdr temp1))
       (section-temp) (device-temp))
      ((null temp1))
      (setq device-temp (car temp1))
    ~check if the device is in another section. If so change
    ~the current section temporarily.
      (cond ((member device-temp (get-all-devices current-section))
             (setq section-temp current-section))
            (t (setq section-temp (get-new-section current-section device-temp))))
      (eval `(change ,section-temp ,device-temp status to ,new-status))
      (eval `(change-bulletin ,section-temp ,device-temp self ,new-status))))
~
(defun get-new-section (section device)
  (cadar (find-value (get section 'inter-equiv) device)))
~
~ ************************************************************
~
~List of files
~
~device:          specifies the characteristics(facts) of a device. The facts
~                 are attached to the property list of the section under
~                 the name of the device.
~change:          changes the value of the facts known.
~find-value:      finds the value of the given slot.
~what-is:         returns the value of the fact presently in the system.
~retrieve:        same as what-is except not a macro.
~add-fact:        apppends additional facts about a particular device.
~is-present-1:    checks the presence of given section, device and slot. If
~                 true returns the slot as specified. Otherwise returns nil.
~is-present-2:    checks the presence of given section and device. If true
~                 returns the facts associated with the device.
~add-value:       add value to the existing slot given the section and device.
~device-description: returns the "frame" or description of the device given.
~get-all-devices: returns all devices attached to the section.

(defmacro device (device-name in section-name facts)
  ~example (device engine IN right-wing (facts))
  ~simply put on the property list of the section.
  `(defprop ,section-name ,facts ,device-name))
~
(defmacro change (section device slot to value)
  ~example (change left-wing airpseed-sensor-1 status to down)
  (prog (temp)
        ~now make sure that the section, device and slot are known
        (cond ((null (setq temp (is-present-1 section device slot))))
              ~if they are make the change.
              (t (return `,(rplaca temp value))))))
~
(defun find-value (facts slot)
  ~example (find-value (specs) status)
  (do ((temp facts (cdr temp))
       (tempcar))
      ((null temp))
      ~set tempcar to the first item in the fact list
      (setq tempcar (car temp))
      (cond ((equal (car tempcar) slot) (return (cdr tempcar))))))
~
```

```
(defmacro what-is (section device slot)
  ~example (what-is left-wing engine linked-to)
  ` ,(car (find-value (get section device) slot)))
~
(defun retrieve (section device slot)
  (car (find-value (get section device) slot)))
~
(defmacro add-fact (section device fact)
  ~example (add-fact tail engine (some useful spec))
  (let ((temp ()))
       ~check if the section and device exist. If yes retrieve the facts
       (cond ((null (setq temp (is-present-2 section device))))
             ~now check if the new fact is already present. If so warn the user
        ((find-value temp (car fact)) (print '(fact already exists. Try change or add-value)))
             ~if alright try add the new fact
             (t ` ,(nconc temp (list fact))))))
~
(defun is-present-1 (section device slot)
  ~call is-present-2 which checks if section and device are present
  (prog (temp)
        (cond ((null (setq temp (is-present-2 section device))))
              ((null (setq temp (find-value temp slot))) (return (print '(no such slot))))
              (t (return temp)))))
~
(defun is-present-2 (section device)
  ~first check if section is present
  (cond ((null (is-section section)) (print '(no such section)))
        ~check if device is known. If so return the facts associated with it.
        ((get section device))
        ~otherwise print warning.
        (t (print '(no such device)))))
~
(defmacro add-value (section device slot value)
  ~example (add-value cockpit airspeed-ind-1 linked-to airspeed-sensor-1)
  (let ((temp ()))
       ~now check if section, device and slot are present. If not print warning
       (cond ((null (setq temp (is-present-1 section device slot))) (print
                          '(value cannot be added to a nonexisting slot)))
             ~if they do then destructively append the new value.
             (t ` ,(nconc (car temp) (list value))))))
~
(defun device-description macro (exp)
  ~example (device-description tail rudder)
  (let ((section (cadr exp))
        (device (caddr exp)))
       ` ,(get section device)))
~
(defun get-all-devices (section)
  ~example (get-all-devices cockpit)
  ~first set templist to entire property list of the section
  (do ((templist (plist section) (cddr templist))
       (answer ()))
      ((null templist) answer)
      ~now form the list containing just the devices.
      (setq answer (cons (car templist) answer))))
~
~ **********************************************************
~
~List of files
~rules:            inserts rules on the property list according to their
~                  rule number.
~similar:          Checks whether two patterns are similar. Has side-effect
~                  of setting variables preceded by ? to proper values.
~check-list:       used by "similar" to check two lists for similarity.
~change-bulletin: command used to make changes on the bulletin.
```

```
~generate-and-test: right now does nothing. Supposed to find if a device can
~                   be salvaged by a different gain or bias.
~look-up:           looks up the entry specified in the database
~restore:           restores the device's condition to normal (or ok)
~same:              compares two values. Return true if they are within 10% of
~                   the larger value.
~add-abnormal-devices: used in monitor mode to add new abnormal alerts on the
~                   bulletin.
~on-abnormal-list: checks if the device given is on the abnormal-devices list.
~
(defmacro rules (rule-number if if-part then then-part)
  ~example (rules rule#1 if (whatever premises) then (whatever conclusions))
  (putprop rule-number if-part 'if)
  `(defprop ,rule-number ,then-part then))
~
(defun similar (pattern match)
  ~example (similar (some stored pattern) (other pattern being matched))
  (do ((pattern-temp pattern (cdr pattern-temp))
       (match-temp match (cdr match-temp)))
      ~stop when nothing is left to be compared
      ((and (null pattern-temp) (null match-temp)) t)
      ~check each element
      (cond ((check-list (car pattern-temp) (car match-temp)))
            (t (return nil)))))
~
(defun check-list (one two)
  ~example (check-list '(1 2 3) '(1 2 (3 4)))
  ~if both are null lists return true.
  (cond ((and (null one) (null two)) t)
        ~otherwise are they identical atoms?
        ((atom one) (cond ((equal one two))
                          ~does the first atom begin with ' . . If so set them =.
                          ((equal '? (car (explode one))) (set one two))
                          ~otherwise return nil
                          (t nil)))
        ~if not atoms recurse on their sub-lists
        (t (and (check-list (car one) (car two))
                (check-list (cdr one) (cdr two))))))
~
(defmacro change-bulletin (section device type new-value)
  ~example (change left-wing airpseed-sensor-1 self 100.0)
  (do ((temp-bulletin bulletin (cddr temp-bulletin)))
      ((null temp-bulletin) (print '(no such section name)))
      ~first match the section
      (cond ((equal section (car temp-bulletin))
             ~if found match the device
             (return (do ((temp-device (cadr temp-bulletin) (cdr temp-device)))
                         ((null temp-device) (print '(no such device name)))
                         (cond ((equal device (caar temp-device))
                           ~if found match the type
                                (return
                                 (do ((temp-type (cdar temp-device) (cddr temp-type)))
                                     ((null temp-type) (print '(no such type name)))
                                     (cond ((equal type (car temp-type))
                                   ~if found replace the associated value
                                      (return `',(rplaca (cdr temp-type) new-value)))))))))))
~
(defun generate-and-test macro (exp)
  ~do nothing
  t)
~
(defun look-up (section-name device-name type)
  (do ((news bulletin (cddr news))
       (temp))
      ~if no entry found, return nil
```

```
        ((null news) nil)
        ~otherwise look for it!
        (cond ((equal (car news) section-name)
               (return (do ((device-list (cadr news) (cdr device-list)))
                           ~again if the entry is not found, return nil
                           ((null device-list) (return nil))
                           ~same section, find the matching device name!
                           (cond ((equal (get-d-name (car device-list)) device-name)
                                  ~Found it! return the value. Otherwise do nothing.
                                  (return (cadr (member type (car device-list))))))))))))
~
(defun restore (section-device)
  ~first remove the device from the abnormal-devices list
  (do ((device-list (cadr (member 'abnormal-devices bulletin)) (cdr device-list)))
      ~if the device is not found, do nothing
      ((null device-list))
      ~Otherwise set the entry to nil which will be removed later by a
      ~garbage-collector-like procedure.
      (cond ((equal section-device (get-abnormal-?device (car device-list)))
             (rplaca device-list ()))))
  ~now change the status of the device in the bulletin and in the database.
  (mark-device (car section-device) (cadr section-device) 'ok))
~
(defun same (value-1 value-2)
  ~if equal then they are certainly the same.
  (cond ((equal value-1 value-2) t)
        ~otherwise it better be within 10% of the larger one.
        ((greaterp value-1 value-2)
         (greaterp 0.1 (quotient (diff value-1 value-2) value-1)))
        (t (greaterp 0.1 (quotient (diff value-2 value-1) value-2)))))
~
(defmacro add-abnormal-devices (section device status-quo)
  (prog (device-list)
        (setq device-list (cadr (member 'abnormal-devices bulletin)))
        (return `(,nconc device-list (list (list section device status-quo))))))
~
(defun on-abnormal-list (section-device)
  (do ((a-list (cadr (member 'abnormal-devices bulletin)) (cdr a-list)))
      ~if it is not on the abnormal-devices list, return nil
      ((null a-list) nil)
      ~check each entry for equality. Return true if found, otherwise keep looking.
      (cond ((null (car a-list)))
            ((equal (get-abnormal-?device (car a-list)) section-device)
             (return t)))))
~


~
```

DTIC

FILMED

4-86

END