Ames

59 p.

NASA Technical Memorandum 88234

IN-14797

(NASA-TM-88234) TIME-BASED AIR TRAFFIC MANAGEMENT USING EXPERT SYSTEMS (NASA) 59 p HC A04/MF A01 CSCL 07G

N86-28068

Unclas

G3/03 43477

Time-Based Air Traffic Management Using Expert Systems

L. Tobias and J.L. Scoggins

April 1986

National Aeronautics and Space Administration

NASA Technical Memorandum 88234

Time-Based Air Traffic Management Using Expert Systems

L. Tobias,
J. L. Scoggins, Ames Research Center, Moffett Field, California

April 1986

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

SUMMARY

A prototype expert system has been developed for the time scheduling of aircraft into the terminal area. The three functions of the air-traffic-control schedule advisor are as follows: First, for each new arrival, it develops an admissible flight plan for that aircraft. Second, as the aircraft progresses through the terminal area, it monitors deviations from the aircraft's flight plan and provides advisories to return the aircraft to its assigned schedule. Third, if major disruptions such as missed approaches occur, it develops a revised plan. The advisor is operational on a Symbolics 3600, and is programmed in MRS (a logic programming language), Lisp, and Fortran.

INTRODUCTION

At present, the decision-making process of air traffic control (ATC) in the extended terminal area (from 150 n. mi. from touchdown until touchdown) is a manual one. Controllers expertly direct traffic flow to maximize capacity without compromising safety by issuing to each aircraft clearances which specify speed, altitude, or heading requirements. In terms of arrivals to a single runway at a major terminal area, this is generally accomplished as follows: along each of the three to four major arrival directions, traffic is spaced uniformly. However, the traffic flows from the various arrival directions are uncoordinated. It is the responsibility of the final controller to take these uncoordinated flows and establish a sequence of traffic which does not violate separation constraints between consecutive aircraft.

The automation of selected ATC decision-making functions using a time-based framework has been the subject of a series of real-time simulation studies (1). The goals of these studies have been to increase landing rate and to reduce fuel consumption, and in addition, to reduce controller workload. Time scheduling is to be accomplished by developing a time plan, and a means of controlling aircraft to achieve a desired time, so that traffic flows which reach the final controller are coordinated, and that little maneuvering is required to achieve the final sequencing.

The key element in the time-based system is a four-dimensional (4D) algorithm (2) which, given the present position, route, and touchdown time, generates a timetable of commands to control the aircraft to follow the route specified and touchdown at the specified time. Flight tests of aircraft equipped with the 4D software have demonstrated that accuracies of ±5 sec can be obtained. In addition, it has been demonstrated in simulation studies (3) that modifications of the 4D algorithms can be used by the ATC computer to generate a speed advisory to assist in achieving time accuracies of ±20 sec for aircraft which are unequipped.[1]

Real-time simulation studies have used these 4D algorithms to investigate the feasibility of a time-based set of candidate operational procedures for handling a mix of 4D-equipped and -unequipped aircraft in the terminal area. The operational procedures include a time-assignment procedure for new arrivals, a procedure to correct for time errors as the aircraft progresses through the terminal area, and a means of handling missed approaches. Results indicate that the procedures investigated are promising; however, they represent only a limited (but important) subset of the procedures which will be required for implementing a time-based system. Attention needs to be focused on a more complete, adaptable plan for scheduling traffic into the terminal area.

This paper describes the initial step in implementing a more complete, time-based system for traffic management in the extended terminal area using the framework of expert systems. A prototype schedule advisor formulated as an expert system is described. First, the rationale for an expert systems approach is given. This is followed by a detailed description of the ATC problem. The software design is then outlined, with a complete, documented code given in the appendix. Finally, plans for the continued development of the schedule advisor are discussed.

RATIONALE FOR AN EXPERT SYSTEMS APPROACH FOR A SCHEDULE ADVISOR

The plan for implementing a schedule advisor in an actual operating environment will be initially to provide a system with limited capabilities, and to gradually increase the sophistication of the computer software as more of the controller's expertise is understood. Thus, the controller would initially possess considerable knowledge not resident in the advisor, and would need to interact with the advisor to modify or override the computer-generated plans. The controller may want to prohibit the scheduling of aircraft during a time interval because of runway visibility problems, to assign priority to one aircraft because it is low on fuel, or to request a particular landing order for a group of aircraft to increase the landing rate.

In addition to the interaction requirement, it is also necessary for the advisor to explain its actions to the controller.[2] The controllers may initially be skeptical of the computer-generated plans, particularly if the computer solution is

[1]Without speed advisories, time accuracies of ±2 min have been achieved by controllers in the en route area.
[2]The explanation capability of expert systems in the ATC context is discussed in Ref. 4.

not in complete agreement with their own intuitive plan. If the logic of the computer plan is understood, the controller will gain confidence in the system more quickly.

It is also expected that the initial set of heuristics will need to be modified. The new heuristics can readily be formulated as added or modified rules: for example, the controller may wish to add a rule to restrict the speed of an aircraft which is in a given altitude range of a given sector under specified traffic flow conditions. It is important that the modifications described be readily implemented.

It is for these primary reasons--the need for a software system which is interactive, which can explain its behavior, and which can easily accept procedural modifications--that an expert systems approach has been formulated. In addition, it should also be noted that there are many other suggested expert system guidelines (5) that are met for this problem area. There are many domain experts that understand and operate the present ATC system, and there are some whose expertise has been built over a long period of time who can commit a substantial amount of time to the development of the system. In addition, the problem domain is well bounded--the problem under consideration is restricted to routes in a major terminal area from 120 n. mi. from touchdown until touchdown. At any one time the number of aircraft in the plan is less than 100, and thus reasoning about aircraft schedules in real time should not be a problem.

THE PROTOTYPE SCHEDULE ADVISOR: THE ATC PROBLEM

A prototype ATC Schedule Advisor has been developed, and will now be described. First, the ATC problem will be presented, followed by a discussion of the system design.

A plan for new arrivals

The terminal area under consideration, shown in Fig. 1, is based on the route structure in a 150 n. mi. radius of Denver, Colorado. The Denver route structure is a "four-corner-post" system with conventional jet aircraft arriving from either Drako from the northwest, Keann from the northeast, Kiowa from the southeast, and Byson from the southwest. A route for low-speed, general-aviation traffic which emanates from Colorado Springs is also considered. It is assumed that Instrument Flight Rules (IFR) prevail, and that all landings are at the Denver Stapleton Airport on runway 26L. The area under consideration contains the Denver Terminal Radar Approach Control (TRACON) and a portion of the Denver Air Route Traffic Control Center (ARTCC).

Conventional jet traffic enters the terminal area region at one of the feeder fixes of the four corner posts described, and general-aviation traffic enters from Colorado Springs. When the traffic enters the terminal area, a time assignment is generated for the aircraft to reach touchdown. The time assignment is based on the range of feasible times for the aircraft to fly the route. The aircraft has a preferred nominal time, but can speed up or slow down within a specified range.

There is an additional FAA requirement in which consecutive aircraft must be separated by a minimum distance; this distance is a function of the weight categories of the aircraft. Three categories have been defined: light, large, and heavy. For the route system under consideration, light aircraft are restricted to the Colorado Springs route, while both large and heavy aircraft can fly on the four jet routes. These distance-separation requirements can be translated into minimum separation times for the purpose of time scheduling (1). Thus, scheduling new arrivals involves both determining the range of feasible times for the aircraft and checking that the proposed schedule does not conflict with previously scheduled aircraft. In the prototype design, the schedule of aircraft which have arrived earlier are considered fixed, and the plan for a new arrival is based on meeting separation constraints with respect to these schedules. It may be possible that no conflict-free time exists. In this case, the new arrival is scheduled for later departure and will hold at the route entry point (feeder fix) until its assigned time.

A typical situation is shown in Fig. 2, which is a time-line representation of all the aircraft currently in the system at time 12:37:03. A separate line is drawn for each of the arrival routes previously discussed, and there is also a line for the missed-approach route, which will be discussed. On the Keann route, two aircraft are shown which have been scheduled; PA003, for which time at touchdown is 12:46:10, and UA003, which directly follows it at 12:49:05. Suppose that at time 12:37:03, WA374 reaches the feeder fix at Drako. The following times are computed by the planner: nominal time to touchdown is 12:52:38, with a range of 12:51:38 to 12:54:38. Any time assignment outside this range would necessitate some holding at Drako. The planner first attempts to schedule WA374 at the nominal, then tries other times on either side of the nominal until the range limits are reached. In this example, the aircraft can be assigned at its nominal time, and no initial holding is necessary. Holding is permitted at some interior points in the region, but interior holding is not part of the initial schedule plan.

Correcting minor schedule deviations

After creating an initial schedule plan, the next function of the schedule advisor is to monitor each aircraft as it proceeds from feeder fix to touchdown, detect deviations from its assigned 4D route, and suggest corrective actions. It is important to note that, while advisories can perhaps be generated continuously in real-time operation, the number of corrective actions (i.e., clearances based on computer-generated advisories) must be kept small. There is still a substantial research problem to optimize the number of advisories, the type of advisories, and the times in which they should be given to the aircraft to minimize the deviation from scheduled touchdown time. One time of advisory, the speed advisory, has been mentioned earlier. In the prototype schedule advisor, the only advisories used are heading and path advisories, which will now be discussed.

When the aircraft departs the feeder fix, a planned touchdown time has been established. A 4D route is synthesized and used as the reference trajectory as the aircraft proceeds from feeder fix to touchdown. To correct for minor deviations from the 4D route, a decision point has been established on each of the routes. Figure 3 shows the decision point for the two northern routes. When the aircraft reaches the decision point D_K on Keann, or D_D on Drako, a path correction is generated so that the aircraft can meet its assigned time schedule. From Drako, the aircraft will be required to adjust its time to turn, while from Keann, a heading change will be required. Note that the advisory will include a warning if, even with the largest correction allowed, an error remains. In the prototype system, no attempt is made to resolve this type of problem. At present, it is necessary for the controller to call for a missed approach, or take some other action to resolve the problem.

Schedule revisions

So far, only minor deviations have been considered, and the strategy has been to generate corrections to return the aircraft to its assigned time so that the schedule plan for other aircraft need not be altered. However, the schedule advisor must consider the situation when major disruptions occur, i.e., when it is not possible to retain the existing plan. These changes may be necessitated by aircraft executing a missed approach or an emergency landing, or by weather-related modifications such as rerouting to avoid severe weather, runway closure, or runway use changes caused by shifting winds. A revised plan must be generated which must handle the disruption in a continually safe manner, and which will return to an expeditious flow after an initial transient period. In the prototype system, two classes of major disruptions are considered: missed approaches and runway closures.

If an aircraft is to execute a missed approach (Fig. 4), it proceeds along its route in level flight toward the touchdown point, then to the departure end of the runway, and then turns right heading toward point H_D where it rejoins the Drako route for a second attempt at a flight to touchdown. A new touchdown time must be assigned for this aircraft, and it is preferred that the missed-approach aircraft be given priority to land at its earliest possible time. This may necessitate holding other aircraft at the feeder fixes or the internal holding points until a revised conflict-free touchdown time is available.

A sample holding situation is illustrated in Fig. 5. Figure 5a shows a time-line plot at time 13:06:54. At this time, the controller decides that AA251 needs to execute a missed approach. The missed-approach command is entered into schedule advisor, and the revised time line shown in Fig. 5b is generated. The aircraft AA251 now appears on the missed-approach line, and has been rescheduled without holding.

Elements of a plan for handling runway closure have been developed. Suppose that the controller declares that a runway is closed for the interval (t_i, t_f), where t_i is the time when the closure is first effective, and t_f is the time when the runway is to be reopened.[3] The advisor first determines which flights are currently assigned in the blocked interval. These flights and any flights which follow them are sequentially rescheduled. Any flight which has passed the last holding point will be told to execute a missed approach and proceed in for a new approach or for holding as required.

A sample runway closure situation is shown in Fig. 6. Figure 6a shows a time line plot at 13:20:16. At this time the controller decides that no aircraft can land in the interval from now until 13:23. One aircraft, PA35, is scheduled to land in this interval. The revised plan is shown in Fig. 6b. No aircraft are scheduled in the blocked interval. The aircraft PA35 has executed a missed approach and is now rescheduled without holding.

THE PROTOTYPE SCHEDULE ADVISOR: SOFTWARE DESIGN

A prototype for the ATC schedule advisor has been programmed on a Symbolics 3600 Lisp machine. A fully commented program listing is given in the appendix; this discussion serves to highlight the key features of the software involved. The intention was to combine existing numerical subroutines, capable of 4D planning single flights, with artificial intelligence (AI) planning techniques to produce an intelligent 4D scheduler for a system of flights. In addition, this scheduler will serve as a research tool, with the ability to add new features as they are developed.

Problem representation

A generalized representation of the problem is to treat the terminal area as a directed graph. A sample graph is given in Fig. 7. Each node on the graph represents some physical point such as touchdown or a feeder fix. These nodes are labeled "a" through "k" in the figure. Each directed arc represents a transition in terms of the time required for a flight to move between the two connected nodes. For example, between the feeder fix "j" and the waypoint "f," a flight may have the option to wait at the holding point labeled "h."

Various information is maintained for each arc, such as the distance to the runway, alternate paths, and procedures that must be executed whenever a flight enters or exits the arc. For example, if a given arc represents the travel between a decision point and the runway, the arc's entry procedure would then include a code to generate a path correction.

Two ordered lists are also maintained for each arc which denote, respectively, the flights that are scheduled to enter and exit the arc. In Fig. 7, the flight PA35 is scheduled to enter at 13:02:23 and then exit at 13:11:45. Upon exiting, PA35 should be flying at an altitude of 36,000 ft and a calibrated airspeed of

[3]The case when the runway is closed indefinitely is more complex and is not considered in the prototype system.

182 knots. Likewise, the flight AA24 has already entered the arc at 12:50:33 and is scheduled to exit at 12:59:57.

This information shows the planned state of the arc at any given time, and can be used to help reschedule flights and to detect potential conflicts. Also, the system can use the list of entering flights at the touchdown arc to determine possible "time slots" where aircraft can be allowed to land. Given this representation, the problem is to develop a planner which finds good time-based paths through the graph for flights that need to be scheduled, and to develop a control structure which operates on the graph as events occur (i.e., flights enter and exit arcs, plans are revised, etc.).

Planning functions

The first planning goal is to always have a consistent plan for the overall system. Each flight in the system must always have a scheduled path from its current position to touchdown. This path consists of a list of scheduled waypoint arrival times which cannot conflict with blocked-runway time intervals or separation requirements for other flights. If situations arise in which the system cannot adequately plan for all of the arriving flights, some of the air traffic will then be placed in holding patterns. For example, if the runway is blocked and a number of flights arrive at the feeder fixes, then some of the flights may have to hold until conflict-free touchdown times can be scheduled.

Next, consider what kind of information the planner must understand and maintain for each flight. This is determined by the numerical routines for 4D path generation and control which lie at the heart of the planner. Input parameters to these routines consist of current time, altitude, calibrated airspeed, distance to touchdown, and speed profile. Output parameters from these routines consist of a list of the computed altitude, airspeed, and distance for the flight at various time steps to touchdown, and the controller advisories required to execute the plan.

The planner must determine a good strategy for a given flight, feed the correct parameters to the numerical routines, and check the resulting plan for any conflicts with flights that are already scheduled in the system. The strategy determines whether a flight should hold at the feeder fix, or use a faster or slower speed profile, etc., and hence determines the parameters to the numerical routines. Plan segments that could be added to arcs on the graph are interpolated from the time steps of the computed flight parameters. Then the planner checks for separation conflicts, blocked-runway conflicts, etc., to test the candidate plan along its path through the graph. This plan is returned if valid; otherwise, the planner backtracks to attempt some alternate strategy.

The planner therefore uses a "generate and test" structure, backtracking whenever some conflict is found. Logic programming is used to encode the planner since backtracking is an inherent feature. Flight strategies can also be represented symbolically and reasoned about conveniently. Heuristics can be added to guide the backtracking. As more engineering knowledge is applied to the ATC problem, more expertise will be encoded into these heuristics to make the path-generation process more efficient.

System architecture

The schedule advisor runs in the Symbolics Zetalisp environment using software programmed in MRS, Lisp, and Fortran. The Lisp functions call a Fortran program to run when the numerical subroutines are needed, and parameters are communicated through input and output data files. The numerical subroutines for 4D path generation and control consist of nearly 3000 lines of a Fortran 77 code originally developed under VAX VMS (3). This code performs a Runge-Kutta integration to determine optimal descent paths. The Symbolics software provides a Fortran 77 development environment and only a few minor changes were needed to port these routines. For instance, the Symbolics Fortran had a shorter line length, and exclamation points could not be used for in-line comments.

Combining Lisp and Fortran code into the same software system produces some challenges. Fortran requires a main program to run before any subroutines can be called, which restricts the use of Fortran subroutines within Lisp programs. To avoid this, the software can be run as a Fortran main program with Lisp functions called as subroutines. Fortunately, data types such as characters, integers, and reals tend to be machine-independent; hence, parameter passing does not require any coercion. Another option chosen for this system is to run Fortran routines as separate programs, or even separate processes, and have Lisp and Fortran communicate through data files or input/output buffers.

One interesting point is that Symbolics Fortran uses Lisp as the intermediate code during compilation, and that it retains the user-defined Fortran variable names for the corresponding Lisp symbols in the intermediate code. This is unfortunate since Fortran programmers tend to use the real variable "T" to denote time values, but the name "T" in Lisp denotes a system-wide symbol for truth. For this system, it was sufficient to rename all Fortran "T" variables as "T1."

Most of the general programming has been done in the Symbolics Lisp language called Zetalisp, which is very close to Common Lisp. This has proven to be a very powerful and well-designed programming language. One major feature is that Zetalisp has a good set of software tools available for writing and debugging code, such as the ZMACS screen editor.

Zetalisp also provides several useful packages, such as TV for windows and graphics, and Flavors for object-oriented programming. The user interface, programmed using the TV package, consists of three windows and various menus as shown in Fig. 8. A "Time Routes" window shows a graphic depiction of the time-scheduled flights in the terminal area. An "Advisories" window displays advisories and expectations about the descending flights. A "Listener" window gives the user access into the system by providing a command prompt and a mouse with sensitive pop-up menus.

Objects, defined with the Flavors package, are useful for representing data structures and for representing knowledge about the system which changes frequently or involves "inheritance." For example, an object is created for each flight that enters the system. Various slots are defined for each flight object, such as "touchdown time," or "neighboring flights." Inheritance is useful in defining classes of objects. For instance, a general class of objects called PLANE could have attributes such as "scheduled touchdown time" "neighboring flights," which all aircraft must have values for. Another type of objects could be defined for HEAVY-AIRCRAFT. As a subclass of PLANE, these would inherit all the properties of PLANE objects, and possibly add other more specific attributes, such as "separation requirements for heavy aircraft."

Other forms of knowledge about the system are represented in MRS, which provides logic programming similar to the Prolog language. MRS, which stands for Meta-level Reasoning System, was developed by Genesereth (6).[4] Logic programming is generally more straightforward than algorithmic methods, so long as the problem can be formulated efficiently in predicate logic. Essentially, the steps are to 1) encode the specifications for a given problem solution, 2) assert the relevant facts into the knowledge base, and 3) query a theorem prover to determine some specific solution. One advantage of formulating ATC planning in logic programming is that the rules can be easily modified and amended as the expertise involved in air traffic control becomes better understood. Another advantage is that the resulting facts and rules can be viewed as the specifications for an optimized version of the system.

MRS has features for solving problems with backward chaining, forward chaining, condition-action rules, and more. In addition, MRS automatically backtracks to search for solutions, and meta-level rules can be defined to order this backtracking (i.e., to incorporate heuristic searches). The planning functions required in this system rely on heuristic techniques since a real-time advisor cannot afford to spend very much time generating alternative 4D paths.

Facts which do not change over time, such as "Flight XX007 executed a missed approach at time 11:30:45," or "Waypoint WP3 is a feeder fix for the KIOWA route," are stored in the MRS knowledge base, along with the various rules. To handle a new arriving flight, the rules determine a good, feasible flight plan in the following order:

1. Use heuristics to develop a possible strategy

2. Call numeric subroutines to compute a plan from this strategy

3. Check constraint rules to test the plan for violations, blocks

---

[4]An earlier version of the schedule advisor by B. Boesch, D. Gregory, J. L. Scoggins, and L. Tobias was implemented entirely in MRS as a class project in Building Expert Systems at Stanford University.

Failure to satisfy the constraint rules at any point will invoke backtracking to search for an alternative plan.

Program operation

Assume that the schedule advisor software runs on a personal workstation networked to host computers for real-time data access. Actual operation of the system basically differs from simulation by the source of the data. Using the Symbolics interface facilities, which provide multiple graphics windows and user interaction menus, the user can create new arrivals, block runways, alter the data structures and knowledge base, execute scenario data files, and step or reset the system clock.

The main program executes a loop to

1. Look for and process input data

2. Update the system clock

3. Check the graph for events that should have occurred by the new current clock time and execute them

At any time, the user can activate a mouse button to interrupt the main loop and request a command menu. Most of the commands are used for data input, such as creating new arrivals, signaling a missed approach, or blocking the runway for some time period. Some other commands provide system controls, such as regulating the Lisp machine's memory garbage collection, or resetting the system clock.

To create a new arrival, for instance, the user selects the "New Arrival" menu option. A special menu then prompts for the flight identification number, type of aircraft, arrival feeder fix, estimated arrival time at the feeder fix, etc. A new object is created to represent this flight, and the planning begins. The resulting advisories are displayed, and the aircraft symbol is plotted on the time line.

CONCLUDING REMARKS

A prototype schedule advisor formulated as an expert system has been developed to assist the air traffic controller in the time scheduling of traffic in the extended terminal area. The advisor develops a time plan for all new arrivals into the terminal area, which takes into account the aircraft performance requirements and the schedules of other aircraft which have already entered into the terminal area. The advisor monitors aircraft flight progress through the terminal area and generates advisories (candidate ATC clearances) to the controller to correct deviations from the time plan. Some initial procedures for revising the plan to accommodate missed approaches and blocked runways have been included.

Future research will be focused on refining the advisor, i.e., applying knowledge of engineering techniques to develop better planning heuristics. The rules developed so far basically demonstrate what types of reasoning can be done, without attempting to be detailed, or to be very precise. This knowledge engineering would depend on a better understanding of the problem domain through:

1. ATC expertise

2. NASA research through simulation, field testing, etc.

3. Encoded domain policies and conventions, e.g., FAA regulations

Among the general issues that need to be addressed in refining the schedule advisor are flight-path optimization, procedures to minimize time deviations at key schedule points, and strategies to improve airport capacity. For example, when planning new arrivals, one question to address would be whether to hold a flight for some time and have it follow a fast speed profile, or to have it just follow a slow speed profile without holding. Another question would be to switch the landing order of scheduled flights as new flights arrive.

As far as the system architecture is concerned, the next step would be to distribute the advisor. For example, it would be good to have both the numerical subroutines and the graphic display windows run as spearate processes. This could alleviate some of the problems found in combining various programming languages in the same run-time environment.

The overall system should be represented in terms of standard data structures, such as directed graph, priority queues, etc. The logic programming would then be used exclusively for the planner; that is, to generate and test strategies and represent search heuristics.

References

(1) Tobias, L., Erzberger, H., Lee, H.Q. and O'Brien, P.J.: "Mixing Four-Dimensional Equipped and Unequipped Aircraft in the Terminal Area," AIAA Journal of Guidance, Control and Dynamics, Vol. 8, No. 3, p. 296: May-June 1985.

(2) Lee, H.Q. and Erzberger, H.: "Time-Controlled Descent Guidance Algorithm for Simulation of Advanced ATC Systems," NASA TM-84373, 1983.

(3) Erzberger, H. and Chapel, J.D.: "Ground Based Concept for Time Control of Aircraft Entering the Terminal Area," Proceedings AIAA Guidance and Control Conference, Snowmass, CO, Aug. 1985.

(4) Cross, S.E.: "Model-Based Reasoning in Expert Systems: An Application to Enroute Air Traffic Control," Proceedings AIAA/IEEE Sixth Digital Avionics Systems Conference, Baltimore, MD, Dec. 1984.

(5) Prerau, D.S.: "Selection of an Appropriate Domain," AI Magazine, Vol. 7, No. 2, p. 18: Summer 1985.

(6) Genesereth, M.R. and Ginsberg, M.L., "Logic Programming," Communications of the ACM, Vol. 28, No. 9, p. 933: Sept. 1985.

APPENDIX

PROGRAM LISTING

```
;;; -*- Mode: LISP; Package: MRS; Syntax: Zetalisp; Base: 10 -*-
;;;
;;; NB: Change this first line and all life as you know it will cease to exist...
;;;
;;;
;;;                          John Scoggins
;;;                     NASA/Ames Research Center
;;;                            M/S 210-9
;;;                      Moffett Field, CA  94035
;;;                          (415) 694-5431
;;;
;;; Overview
;;;
;;;  This file contains the top level definitions and system calls to start the
;;; ATC Scheduler expert system.  Simply use the command:
;;;
;;;     Load File CHA:>Scoggins>newaveo>atc
;;;
;;;  This file will then:
;;;
;;;      • Add the ATC system as a select option on the lisp machine
;;;      • Define the ATC system in terms of file dependencies
;;;      • Compile and load the ATC system
;;;      • Run the command loop to start the system
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; Define the system in terms of the required files and which
;compilers they use

(defsystem ATC-Scheduler

  (:name "NASA Air Traffic Control Scheduler")  ;System names and defaults
  (:short-name "ATC Scheduler")
  (:pathname-default "CHA:>Scoggins>newaveo>")
  (:package MRS)

  (:module fonts   ("atcfont"))                 ;Module names and files
  (:module pathgen ("pathgen"))
  (:module objects ("user" "graph"))
  (:module planner ("planner"))

  (:load-bfd fonts)                             ;Compiler dependencies
  (:fortran-compile-load pathgen)
  (:compile-load objects)
  (:readfile planner)
  )


; Compile and load the system

(make-system 'ATC-Scheduler
             :compile
             :noconfirm
             )


; Call the command loop for FRAME to run the system

(send FRAME :command-loop)

(Comment (process-run-function
           "ATC Scheduler"
           '(lambda ()
              (send FRAME :command-loop)
              )
           )
         )
```

```
;;; -*- Mode: LISP; Package: MRS; Syntax: Zetalisp; Base: 10 -*-
;;;
;;; NB: Change this first line and all life as you know it will cease to exist...
;;;
;;;
;;;                         John Scoggins
;;;                   NASA/Ames Research Center
;;;                           M/S 210-9
;;;                   Moffett Field, CA  94035
;;;                        (415) 694-5431
;;;
;;; Overview
;;;
;;;  This file contains the MRS rules and facts required for the planning
;;; functions in the ATC Scheduler expert system.
;;;
;;;  This file will:
;;;
;;;     • Create and clear its own MRS theory called ATC
;;;     • Set up the meta level definitions for the knowledge base
;;;     • Stash the rules, facts and demons into the knowledge base
;;;     • Define the planning functions and interface to the FORTRAN code
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Meta Level Definitions
;;;
;;;  These rules, facts and functions make up the systems level incantations
;;; needed to support demons, backward chaining, rule ordering, and
;;; procedural attachment into lisp and the object base.  Logicians use
;;; the term "meta" to describe propositions about an inference process.
;;; Basically, we use the meta level to short-circuit MRS, for example
;;; when we need to access the object base.
;;;
;;;  Contrary to what the manual may say...  we have included any "wisdom"
;;; about the MRS system that we had to learn the hard way, ie. there
;;; are bugs and contradictions to the documentation.
;;;
;;; The propositions DONE and CUT will only work at the meta level
;;; and not at the base level.  EVAL has been added to take the place
;;; of DONE at the base level.
;;;
;;; The difference between ELEMENT and MEMBER is that only the latter
;;; will bind its first argument to a list of variables.
;;;
;;; RESIDUE will not support generate and test strategies since it
;;; tries to find all solutions instead of backtracking when necessary.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(setq theory 'GLOBAL)                          ;repair meta level to forward chain
(unstash '(toassert (and . &l) assert-and))    ;with more than one conclusion - this
(stash '(toassert &p fc))                      ;enables explanations for the demons
(stash '(toassert (and . &l) assert-and))

(setq theory 'ATC)                             ;define a theory and clear it each time
(empty theory)                                 ;we reload the file - also load any
; (load "CHA:>mrs>set")                        ;necessary MRS packages, like SET

(stash '(tolookup (EVAL &funct . &args)        ;procedural attachment to call
                  fq-lookup))                  ;LISP and FORTRAN routines from
(stash '(tolookups (EVAL &funct . &args)       ;MRS
                   fq-lookups))
(stash '(lisp EVAL EVAL))


; Short cuts to call the most commonly need LISP functions without using EVAL - see
;the "Complete Guide to MRS" under procedural attachment

(stash '(tolookup (FORMAT . &args) lisp-lookup))
(stash '(tolookups (FORMAT . &args) lisp-lookups))

(stash '(tolookup (GETBDG . &args) lisp-lookup))
(stash '(tolookups (GETBDG . &args) lisp-lookups))

(stash '(tolookup (SEND . &args) lisp-lookup))
(stash '(tolookups (SEND . &args) lisp-lookups))

(stash '(tolookup (SETQ . &args) lisp-lookup))
(stash '(tolookups (SETQ . &args) lisp-lookups))


(Defun LISP-LOOKUP
       (p)                                     ;MRS has FQ-LOOKUPS defined to do this

  (let ((vals (get-term p)))                   ;kind of procedural attachment, but it
    (unifyp                                    ;really only works for the numerical
      (eval                                    ;functions.  LISP-LOOKUP(S) replaces the
        (car vals))                            ;the EVAL proposition defined in RULES.LISP
      (cadr vals)))                            ;This is called a "computable representation"
  )                                            ;see the file CHA:>MRS>efrepn.lisp


(Defun LISP-LOOKUPS
       (p)                                     ;MRS requires a pluralized form for each

  (let ((vals (get-term p)))                   ;computable representation
    (pluralize
      (unifyp
        (eval
          (car vals))
        (cadr vals))))
  )

(Defun LEQP                                          ;MRS defines these comparison operators
       (&rest args)                                  ;for two operands only... this patch allows
                                                     ;us to use queries like:
  (apply '<= args)                                   ; (TRUEP '(<= 2 3 4))
  )

(Defun GEQP
       (&rest args)

  (apply '>= args)
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Utility Rules and Definitions
;;;
;;;  These misfit propositions have nothing to do with either the meta level
;;; or the domain knowledge base.  They just make life easier when using MRS.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(stash '(append nil $lst $lst))                ;Append two list to return a third
                                               ;list as the result - this is an
(stash '(append $lst nil $lst))                ;excellent example of logic programming
                                               ;for any Prologer who wants to learn MRS

(stash '(if (append $cdr $lst $ans)
            (append ($car . $cdr) $lst ($car . $ans))
            ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Planning Functions
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(Defun PLAN-NEW-ARRIVAL
       (flight class route time alt cas)
  ;
  ;Instantiate flight with its parms
  (set flight
       (make-instance 'FLIGHT
                      :label (get-pname flight)
                      :class class
                      :current-route route
                      ))
  ;
  ;Query to find and install plan segments
  (let* ((approach (eval (implode (append '(A P P -) (explode route)))))
         (init-seg (list (eval flight) approach  time time (send approach :dist-to-td) alt cas))
         (query (truep '(planned ,init-seg  $segs $advs)))
         )
    (send (eval flight) :install-plan (getvar '$segs query) (getvar '$advs query))
    )
  )


(Defun PLAN-MISSED-APPROACH
       (flight)
  ;
  ; Find the flight's segment on TD-ARC and move it to MISSED
  (let* ((td-seg (send flight :edit-plan-at TD-ARC))
         (miss-seg (list flight MISSED (nth 2 td-seg) (nth 3 td-seg) 0.0 36000.0 180.0))
         )
    (send TD-ARC :del-flight td-seg)
    (send flight :install-plan
          (list miss-seg)
          (list (format nil "Flight ~A executes a missed approach at ~\time\"
                        (send flight :label) (nth 2 miss-seg)))
          )
    ;
    ;Query to find and install revised plan segments
    (let* ((query (truep '(planned ,miss-seg $segs $advs)))
           )
      (send flight :install-plan (getvar '$segs query) (getvar '$advs query))
      )
    )
  )


(Defun PLAN-BLOCKED-RUNWAY
       (flight)
  ;
  ; Remove flight plan segments past current arc
  (let* ((oldies (reverse (send flight :segments)))
         (init-cur (send flight :edit-plan-at (send flight :current-arc)))
         (cur-segs (send flight :segments))
         (init-old (list flight MISSED (nth 2 (car oldies)) (nth 2 (car oldies)) 0.0 36000.0 180.0))
         (old-segs (reverse (cons init-old (cdr oldies))))
         )
    ;
    ;Remove all trace of the flight until it is replanned
    (send flight :set-segments oldies)
    (send flight :remove-self)
    (send flight :describe)
    (send TERMINAL-GRAPH :add-flight flight)
    (send flight :set-segments nil)
    ;
    ;Query to find and install revised plan segments
    (let* ((query (truep '(block-replanned ,init-cur ,cur-segs ,init-old ,old-segs $segs $advs)))
           )
      (send flight :install-plan (getvar '$segs query) (getvar '$advs query))
      )
    )
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Planning Facts and Rules for Path Generation
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(stash '(if (and (planned $init-old $segs1 $advs1)
                 (append $old-segs $segs1 $segs)
                 (append $advs1 ("Missed approach forced by blocked runway") $advs)
                 )
            (block-replanned $init-cur $cur-segs $init-old $old-segs $segs $advs)
            ))


(stash '(if (and (planned $init-cur $segs1 $advs1)
                 (append $cur-segs $segs1 $segs)
                 (append $advs1 ("Rescheduled due to blocked runway") $advs)
                 )
            (block-replanned $init-cur $cur-segs $init-old $old-segs $segs $advs)
            ))


(stash '(if (and (= $init-seg ($flt $arc $entry $exit $dis $alt $cas))
                 (gen-path $arc ($arc . $path))
                 (gen-segs $init-seg $path $segs $advs)
                 (no-block-conflicts $segs)
                 )
            (planned $init-seg $segs $advs)
            ))


(stash '(if (and (send $arc :next-arcs $next-arcs)
                 (member $next $next-arcs)
                 (gen-path $next $result-path)
                 )
            (gen-path $arc ($arc . $result-path))
            ))


(stash '(gen-path ,TD-ARC (,TD-ARC))
            )


(stash '(if (and (= $init-seg ($flt $arc $entry $exit $dis $alt $cas))
                 (strategy $profile)
                 (eval (PATH-GEN '$profile '$flt '$path '$exit '$alt '$cas)
                   ($segs $advs))
                 (no-flight-conflicts $segs)
                 )
            (gen-segs $init-seg $path $segs $advs)
            ))


(stash '(if (and (send $hold-arc :isa-hold-pt t)
                 (gen-hold $init-seg $hold-arc $hold-seg $hold-adv)
                 (no-flight-conflicts ($hold-seg))
                 (gen-segs $hold-seg $path $segs $advs)
                 )
            (gen-segs $init-seg ($hold-arc . $path) ($hold-seg . $segs) ($hold-adv . $advs))
            ))


(stash '(if (and (= $init-seg ($flt $arc $entry $exit $dis $alt $cas))
                 (send $flt :label $flt-label)
                 (send $hold-arc :label $arc-label)
                 (member $hold (30 60 120 240 480 10000))
                 (+ $exit $hold $hold-exit)
                 (format nil
                   "Flight ~A holds for ~0$ sec at ~0$ ft over the ~A"
                   '$flt-label '$hold '$alt '$arc-label
                   $hold-adv
                   )
                 (= $hold-seg ($flt $hold-arc $exit $hold-exit $dis $alt $cas))
                 )
            (gen-hold $init-seg $hold-arc $hold-seg $hold-adv)
```

```
          ))

(stash '(if (unprovable
              (and (member $this-seg $segs)
                   (= ($flt $arc $entry $exit $dis $alt $cas) $this-seg)
                   (send $arc :will-conflict '$this-seg ($front $back))
                   (+ $front $back $conflict)
                   (> $conflict 0)
                   ))
            (no-flight-conflicts $segs)
            ))

(stash '(if (unprovable
              (and (member ($flt ,TD-ARC $entry $exit $dis $alt $cas) $segs)
                   (blocked-runway $t1 $t2)
                   (< $t1 $entry $t2)
                   ))
            (no-block-conflicts $segs)
            ))

(stash '(strategy FAST)                        ;until FORTRAN is debugged...
       )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Interface to Numerical Subroutines for Path Generation
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(Defun PATH-GEN
       (type flight path time altd spd)
  ;
  (list
    (loop with prev-time = 0.0
          for (arc enter dis alt cas)
              in (reverse (PATH-GEN-AUX type (reverse path) time altd spd))
          for exit first enter then prev-time
          collect (list flight arc enter exit dis alt cas) into segs
          do (setq prev-time enter)
          finally (return segs)
                )
    (cons
      (format nil "Flight ~A:" (send flight :label))
      (with-open-file
        (advs "CHA:>Scoggins>advisories.data" ':direction ':input)
        (loop while (send advs :listen)
              collect (readline advs)
                )
        )
      )
    )
  )


(Defun PATH-GEN-AUX
       (type path time altd spd)
  ;
  ; Find the distances for each arc in the given path
  (setq path
        (loop for arc in (reverse path)
              collect (cons arc (send arc :dist-to-td))
                )
        )
  ;
  ; Write the initial flight parameters to the file "flt-parms.data"
  (with-open-file
    (parms "CHA:>Scoggins>flt-parms.data" :direction :output)
    (format parms
            "~A ~$ ~$ ~$ ~$~%"
            (selectq type                ;map the speed profile parameter value
              (FAST 1) (NOMINAL 2) (SLOW 3))
            (float time)
            (cdar path)
            (max altd 4650.0)            ;error check the alt parameter
            (max spd 154.0)              ;error check the cas parameter
            )
    )
  ;
  (f77:execute PathGen
               :init-to-zero t                          ;Initialize FORTRAN variables to zero
               :pathname-default  "CHA:>Scoggins>"      ;Define logical units for files
               :units ((2  "cmdtab.data")
                       (9  "eprdata3.data")
                       (18 "flt-parms.data")
                       (19 "flt-segs.data")
                       (20 "advisories.data")
                       )
               )
  ;
  (let*
    ((tim (make-array '(100)))
     (dis (make-array '(100)))
     (alt (make-array '(100)))
     (cas (make-array '(100)))
     (nsegs 0)
     (coords nil)
     )
    ;
    (with-open-file
      (segs "CHA:>Scoggins>flt-segs.data" :direction :input)
      (loop for i from 0 below 100
            while (send segs :listen)
            do (aset (read-for-top-level segs) tim i)
            do (aset (read-for-top-level segs) dis i)
            do (aset (read-for-top-level segs) alt i)
            do (aset (read-for-top-level segs) cas i)
            finally (setq nsegs (if (< (aref dis (- i 2)) 0.0)
                                    (- i 2)
                                    (- i 1)
                                    ))
                  )
      )
    ;
    (aset (+ (aref tim (1- nsegs))
             (// (* (aref dis (1- nsegs))
                    3600.0)
                 (aref cas (1- nsegs))
                 ))
          tim
          nsegs)
    ;
    (aset 0.0 dis nsegs)
    (aset 0.0 alt nsegs)
    (aset (aref cas (1- nsegs)) cas nsegs)
    (catch 'PROF-EXIT
      (loop for j from 0 below nsegs
            as d1 = (aref dis j)
            as d2 = (aref dis (1+ j))
            do (loop for (arc . d) in path
                     while (>= d1 d d2)
                     do (push
                          (list arc
                                (+ time          ;time
                                   (pt-slope d
                                             d1 (fix (aref tim j))
                                             d2 (fix (aref tim (1+ j))))
                                   )
                                d                ;distance
                                (pt-slope d      ;altitude
                                          d1 (aref alt j)
                                          d2 (aref alt (1+ j))
                                          )
                                (pt-slope d      ;speed
                                          d1 (aref cas j)
                                          d2 (aref cas (1+ j))
                                          )
                                )
                          coords)
                     do (pop path)
                     when (null path) do (throw 'PROF-EXIT (reverse coords))
                                          )
                )
      )
    )
  )


(Defun PT-SLOPE                                ;use the point-slope formula to
       (x x1 y1 x2 y2)                         ;linearly interpolate between two points

  (cond ((zerop (- x2 x1))
         'Undefined
         )
        ((floatp y1)
         (+ y1
            (* (- x x1)
               (// (- y2 y1)
                   (- x2 x1))))
         )
        (t
         (+ y1
            (fix (* (- x x1)
                    (// (float (- y2 y1))
                        (- x2 x1)))))
        )
       )
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Planning Facts and Rules for Path Correction
;;;
;;;
;;;  For now, we just use a uniformly distributed random variable to simulate the
;;; amount of path error.
;;;
;;;  This should even be smarter, ie. if a flight is too early, but there is no
;;; runway block or time separation conflict in front of its scheduled touchdown,
;;; then why bother with path stretching?
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; CORRECTION determines how a flight should maneuver to correct, based on the
;calculated time descrepancy.

(stash '(if (and (- $tdly $delta $error)
                 (format nil
                   "Flight ~A executes ~A ~2$ late to correct for time error~%WARNING Flight ~A is ~0$ s
ec early even with corrections"
                   '$label '$man '$ddly '$label '$error $msg)
                 )
            (correction $label $t $man $delta $dadv $tadv $ddly $tdly $msg)
            ))


(stash '(if (and (<= $tdly $delta)
                 (* $delta $ddly $temp)
                 (// $temp $tdly $correct)
                 (format nil
                   "Flight ~A executes ~A ~2$ late to correct for time error"
                   '$label '$man '$correct $msg)
                 )
            (correction $label $t $man $delta $dadv $tadv $ddly $tdly $msg)
            ))


(stash '(if (and (<= -5.0 $delta)
                 (format nil
                   "Flight ~A requires no path correction"
                   '$label $msg)
                 )
            (correction $label $t $man $delta $dadv $tadv $ddly $tdly $msg)
            ))


(stash '(if (and (< 5.0 $delta)
                 (* $delta $dadv $temp)
                 (// $temp $tadv $correct)
                 (format nil
                   "Flight ~A executes ~A ~2$ early to correct for time error"
                   '$label '$man '$correct $msg)
                 )
            (correction $label $t $man $delta $dadv $tadv $ddly $tdly $msg)
            ))


(stash '(if (and (< $tadv $delta)
                 (- $delta $tadv $error)
                 (format nil
                   "Flight ~A executes ~A ~2$ early to correct for time error~%WARNING Flight ~A is ~0$
sec late even with corrections"
                   '$label '$man '$dadv '$label '$error $msg)
                 )
            (correction $label $t $man $delta $dadv $tadv $ddly $tdly $msg)
            ))
```

```
;;; -*- Mode: LISP; Package: MRS; Syntax: Zetalisp; Base: 10 -*-
;;;
;;; NB: Change this first line and all life as you know it will cease to exist...
;;;
;;;
;;;                         John Scoggins
;;;                    NASA/Ames Research Center
;;;                           M/S 210-9
;;;                     Moffett Field, CA  94035
;;;                        (415) 694-5431
;;;
;;; Overview
;;;
;;;  This file defines the flavors and methods used in the ...
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(Declare
  (special
    ;Here's a list of all the names for instantiated objects which will be
    ;referred to by these flavors and methods, ie. the "forward references".
    ;
    FRAME
    TO-ARC
    TERMINAL-GRAPH
    )
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; FLIGHT Flavor Definition and Methods
;;;
;;;
;;;     Here's what a flight segment is supposed to look like:
;;;
;;;     (FLIGHT ARC ENTER-TIME EXIT-TIME DIS ALT CAS)
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(DefFlavor FLIGHT
        ;This is the generic flight flavor to represent aircraft in the
        ;system.
        ;
        ((class nil)                          ;aircraft size class
         (current-arc nil)                    ;current position
         (current-route nil)                  ;current route
         (advisories nil)                     ;advisories list
         (segments nil)                       ;plan segment list
         (label nil)                          ;printed label string
         (icon #/a)                           ;graphics character
         )
        ;
        ()
  ;
  :gettable-instance-variables                ;these flags mean that any of the
  :settable-instance-variables                ;flavor's slot can be initialized,
  :initable-instance-variables                ;accessed or reset
  )


(DefMethod (FLIGHT :after :init)
           (ignore)
  ;This demon adds every newly instantiated flight to the TERMINAL-GRAPH's
  ;list of flights.
  ;
  (setq icon (if (eq class 'small) #/b #/a))
  (send TERMINAL-GRAPH :add-flight self)
  )


(DefMethod (FLIGHT :install-plan)
           (new-segs new-advs)
  (setq advisories (append advisories new-advs))
  (if new-advs (send FRAME :print-advisories new-advs))
  (setq segments (sort (append segments new-segs)
                       '(lambda (seg1 seg2)
                          (< (nth 2 seg1) (nth 2 seg2))
                          ))
           )
  (loop for seg in new-segs
        do (send (nth 1 seg) :add-flight seg)
           )
  )


(DefMethod (FLIGHT :edit-plan-at)
           (last-arc)
  ;
  (loop for seg in segments
        while (neq last-arc (nth 1 seg)) collect seg into good-segs
        finally (setq segments good-segs)
        finally (return seg)
                )
  )


(DefMethod (FLIGHT :time-sep)
           (other-flight)
  ;
  (cadr (assoc (cons class (send other-flight :class))
               '(((HEAVY . HEAVY) 94) ((HEAVY . LARGE) 74) ((HEAVY . SMALL) 74)
                 ((LARGE . HEAVY) 114) ((LARGE . LARGE) 74) ((LARGE . SMALL) 74)
                 ((SMALL . HEAVY) 167) ((SMALL . LARGE) 38) ((SMALL . SMALL) 98))
```

```
              ))
  )


(DefMethod (FLIGHT :report-plan)
           ()
  ;
  (loop for (flight arc enter exit dis alt cas) in (reverse segments)
        initially (format t "~%Flight ~A is scheduled as follows:~%" label)
        do (format t "~%~\time\ ~A" enter (send arc :label))
           )
  )


(DefMethod (FLIGHT :remove-self)
           ()
  ;
  (loop for seg in segments
        for arc = (nth 1 seg)
        do (send arc :del-flight seg)
           )
  ;
  (send TERMINAL-GRAPH :del-flight self)
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  ARC Flavor Definition and Methods
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(DefFlavor ARC
        ;This is the generic arc flavor to represent an arc between
        ;waypoints in the terminal area.

        ((entering-flights nil)                 ;entering flight segments
         (exiting-flights nil)                  ;exiting flight segments
         (isa-hold-pt nil)                      ;flag for holding arcs
         (from-node nil)                        ;source waypoint
         (to-node nil)                          ;destination waypoint
         (dist-to-td 0.0)                       ;distance to touchdown
         (next-arcs nil)                        ;connecting arcs
         (parent-route nil)                     ;parent route
         (label nil)                            ;printed label string
         (symbol nil)                           ;object symbol name
         )
        ;
        ()
  ;
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables
  )


(DefMethod (ARC :after :init)
           (ignore)
  ;This daemon adds every newly instantiated arc to the TERMINAL-GRAPH's
  ;list of flights.
  ;
  (send TERMINAL-GRAPH :add-arc self)
  )


(DefMethod (ARC :add-flight)
           (flt-seg)
  ;This method inserts a new flight plan segment "flt-seg" into the list
  ;of flight segments in sorted order.
  ;
  (if (≥ (nth 3 flt-seg) (send FRAME :time))
      (setq entering-flights
            (sort (cons flt-seg entering-flights)
                  '(lambda (seg1 seg2)
                     (< (nth 2 seg1) (nth 2 seg2)))
                  ))
      )
  )


(DefMethod (ARC :del-flight)
           (flt-seg)
  ;This method deletes the given flight plan segment "flt-seg" from the list
  ;of flight segments.
  ;
  (setq entering-flights (remove flt-seg entering-flights)
        exiting-flights (remove flt-seg exiting-flights)
        )
  )


(DefMethod (ARC :reset)
           ()
  ;This method removes all the old flight plans to reset the system state before
  ;a scenario file is restored.
  (setq entering-flights nil
        exiting-flights nil
        )
  )
```

```
(DefMethod (ARC :run-entry-code)
           (flt-seg time)
  ;
  (send FRAME :print-advisories
        (list (format nil "Flight ~A should have reached the ~A by ~\time\"
                      (send (car flt-seg) :label)
                      label
                      time
                      )))

  (send (car flt-seg) :set-current-arc self)

  (setq entering-flights
        (remove flt-seg entering-flights))

  (setq exiting-flights
        (sort (cons flt-seg exiting-flights)
              '(lambda (seg1 seg2)
                 (< (nth 3 seg1) (nth 3 seg2)))
              ))
  )


(DefMethod (ARC :run-exit-code)
           (flt-seg time)
  ;
  (setq exiting-flights
        (remove flt-seg exiting-flights))
  (+ 1 time)
  )


(DefMethod (ARC :update)
           (time)
  ;
  (loop for flt-seg in entering-flights
        while (≥ time (nth 2 flt-seg))
          do (send self :run-entry-code flt-seg time)   ;execute the arc's entry code
             )
  ;
  (loop for flt-seg in exiting-flights
        while (≥ time (nth 3 flt-seg))
          do (send self :run-exit-code flt-seg time)    ;execute the arc's exit code
             )
  )


(DefMethod (ARC :will-conflict)
           (new-seg)
  ;
  (let* ((fnt-bck
           (loop with front = nil
                 with enter-time = (nth 2 new-seg)
                 for seg in entering-flights
                 when (< enter-time (nth 2 seg))
                   return (cons nil seg)
                 while (≥ enter-time (nth 2 seg))
                 do (setq front seg)
                 finally (return (cons front
                                       (cadr (member front
                                                     entering-flights))))
                        )
           ))
    (list (send self :segs-conflict (cdr fnt-bck) new-seg)
          (send self :segs-conflict new-seg (car fnt-bck)))
    )
  )


(DefMethod (ARC :segs-conflict)
           (seg1 seg2)
  ;
  (if (not (and seg1 seg2))
      0
      (abs (min 0
                (- (nth 2 seg1)
                  (nth 2 seg2)
                  (send (car seg1) :time-sep (car seg2))
                  )))
      )
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Flavor Definitions and Methods for Specific ARC Types
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(DefFlavor APPROACH-ARC
        ()
        (ARC)
  )


(DefFlavor HOLDING-ARC
        ((isa-hold-pt t)                        ;holding flag set true
         )
        (ARC)
  )


(DefMethod (HOLDING-ARC :will-conflict)
           (new-seg)
  ;
  ; Nothing conflicts here yet...
  nil
  )


(DefFlavor CORRECTION-ARC
        ((advance 3.0)
         (delay 4.0)
         (maneuver "turn to base leg")
         )
        (ARC)
  )


(DefMethod (CORRECTION-ARC :after :run-entry-code)
           (flt-seg time)
  ;
  (send (car flt-seg) :install-plan
        nil
        (list
          (getbdg '$msgs
                (list 'correction
                      (send (nth 0 flt-seg) :label)     ;flight label
                      (nth 2 flt-seg)                   ;entering (correction) time
                      maneuver                          ;required maneuver
                      (- (mod (abs (random)) 20) 10)    ;randomly generated error
                      advance                           ;distance of advance allowed
                      (// (* advance 7200.0)            ;time of advance allowed
                          (nth 6 flt-seg))
                      delay                             ;distance of delay allowed
                      (// (* delay -7200.0)             ;time of delay allowed
                          (nth 6 flt-seg))
                      '$msgs                            ;returned correction messages
                      ))
            )
        )
  )


(DefFlavor TOUCHDOWN-ARC
        ()
        (ARC)
  )


(DefMethod (TOUCHDOWN-ARC :block-runway)
           (time1 time2)
  ;
  (loop for (flight arc enter . rest) in entering-flights
        when (≤ time1 enter time2)
          do (PLAN-BLOCKED-RUNWAY flight)
             )
  )
```

```
(DefFlavor MISSED-ARC
        ()
        (ARC)
  )


(DefMethod (MISSED-ARC :after :run-entry-code)
           (flt-seg time)
  ;
  (send (car flt-seg) :set-current-route 'MISSED)
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  GRAPH Flavor Definition and Methods
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(DefFlavor GRAPH
        ;
        ((arc-list nil)
         (flight-list nil)
         )
        ;
        ()
  ;
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables
  )


(DefMethod (GRAPH :add-arc)
           (arc)
  ;
  (setq arc-list (cons arc arc-list))
  )


(DefMethod (GRAPH :del-arc)
           (arc)
  ;
  (setq arc-list (remove arc arc-list))
  )


(DefMethod (GRAPH :add-flight)
           (flight)
  ;
  (setq flight-list (cons flight flight-list))
  )


(DefMethod (GRAPH :del-flight)
           (flight)
  ;
  (setq flight-list (remove flight flight-list))
  )


(DefMethod (GRAPH :update)
           (time)
  ;
  (loop for arc in arc-list
        do (send arc :update time)
           )
  )


(DefMethod (GRAPH :save-scenario)
           (pathname)
  ;
  (with-open-file
    (save pathname :direction :output)
    ;
    ;Write file attribute list
    (format save "; -*- Mode: LISP; Package: MRS; Syntax: Zetalisp; Base: 10 -*-~2%")
    ;
    ;Write the list of runway blocks
    (loop for block in (trueps '(blocked-runway $t1 $t2))
          collect (format save
                          "(stash '(blocked-runway ~A ~A))~2%"
                          (getvar '$t1 block)
                          (getvar '$t2 block)
                          )
           )
    ;
    ;Write the current system time
    (format save
            "(send FRAME :reset-time ~A)~2%"
            (send FRAME :time)
            )
    ;
    ;Write list of flight object instances
    (loop for flt in (reverse flight-list)
          for label = (send flt :label)
          do (format save
                     "(setq ~A (make-instance 'FLIGHT
                                 :label (get-pname '~A)
                                 :class '~A
                                 :current-route '~A
                                 :current-arc ~A
                                 ))~2%"
                     label
                     label
                     (send flt :class)
                     (send flt :current-route)
                     (send (send flt :current-arc) :symbol)
                     )
          do (format save
                     "(send ~A :install-plan '~A nil )~2%"
                     label
                     (loop for (flt2 arc . rest) in (send flt :segments)
                           collect (format nil
                                           "(,~A ,~A . ~A)"
                                           label
                                           (send arc :symbol)
                                           rest
                                           )
                           )
                     )
             )
    )
  )


(DefMethod (GRAPH :restore-scenario)
           (pathname)
  ;
  ;Remove all the old flights and flight plans and empty scratch theory
  (setq flight-list nil)
  (loop for arc in arc-list
        do (send arc :reset)
           )
  (empty theory)
  ;
  ;Read in the given scenario file
  (load pathname)
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Compile, Load Flavors; and Instantiate Objects
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(Compile-Flavor-Methods
  FLIGHT
  ARC APPROACH-ARC HOLDING-ARC CORRECTION-ARC TOUCHDOWN-ARC MISSED-ARC
  GRAPH
  )


(setq TERMINAL-GRAPH
      (make-instance 'GRAPH))


(Comment (init-path Drako  wp6 wp10 wp14 wp20 wp53 wp42 wp35 wp39 D TD)
         (init-path Keann  wp4 wp15 wp17 wp19 wp20 wp53 wp42 wp35 wp39 D TD)
         (init-path COS    wp9 wp45 wp42 wp35 wp39 D TD)
         (init-path Kiowa  wp3 wp22 wp23 wp25 wp54 wp42 wp35 wp39 D TD)
         (init-path Byson  wp7 wp28 wp29 wp25 wp54 wp42 wp35 wp39 D TD)
         (init-path Missed D TD Miss wp53 wp42 D TD)
         )


(setq MISSED
      (make-instance 'MISSED-ARC
                     :dist-to-td 150.0
                     :label "missed approach point"
                     :symbol 'MISSED
                     ))

(setq TD-ARC
      (make-instance 'TOUCHDOWN-ARC
                     :dist-to-td 0.0
                     :label "runway"
                     :symbol 'TD-ARC
                     ))

(setq COR-DRAKO
      (make-instance 'CORRECTION-ARC
                     :dist-to-td 30.0
                     :label "Drako decision point"
                     :symbol 'COR-DRAKO
                     ))

(setq COR-KEANN
      (make-instance 'CORRECTION-ARC
                     :dist-to-td 30.0
                     :label "Keann decision point"
                     :symbol 'COR-KEANN
                     ))

(setq COR-COS
      (make-instance 'CORRECTION-ARC
                     :dist-to-td 30.0
                     :label "COS decision point"
                     :symbol 'COR-COS
                     ))

(setq COR-KIOWA
      (make-instance 'CORRECTION-ARC
                     :dist-to-td 30.0
                     :label "Kiowa decision point"
                     :symbol 'COR-KIOWA
                     ))

(setq COR-BYSON
      (make-instance 'CORRECTION-ARC
                     :dist-to-td 30.0
                     :label "Byson decision point"
                     :symbol 'COR-BYSON
                     ))
```

```
(setq FF-DRAKO
      (make-instance 'ARC
                     :dist-to-td 100.0
                     :label "Drako feeder fix"
                     :symbol 'FF-DRAKO
                     ))

(setq FF-KEANN
      (make-instance 'ARC
                     :dist-to-td 100.0
                     :label "Keann feeder fix"
                     :symbol 'FF-KEANN
                     ))

(setq FF-COS
      (make-instance 'ARC
                     :dist-to-td 100.0
                     :label "COS feeder fix"
                     :symbol 'FF-COS
                     ))

(setq FF-KIOWA
      (make-instance 'ARC
                     :dist-to-td 100.0
                     :label "Kiowa feeder fix"
                     :symbol 'FF-KIOWA
                     ))

(setq FF-BYSON
      (make-instance 'ARC
                     :dist-to-td 100.0
                     :label "Byson feeder fix"
                     :symbol 'FF-BYSON
                     ))

(setq HOLD-DRAKO
      (make-instance 'HOLDING-ARC
                     :dist-to-td 100.0
                     :label "Drako holding point"
                     :symbol 'HOLD-DRAKO
                     ))

(setq HOLD-KEANN
      (make-instance 'HOLDING-ARC
                     :dist-to-td 100.0
                     :label "Keann holding point"
                     :symbol 'HOLD-KEANN
                     ))

(setq HOLD-COS
      (make-instance 'HOLDING-ARC
                     :dist-to-td 100.0
                     :label "COS holding point"
                     :symbol 'HOLD-COS
                     ))

(setq HOLD-KIOWA
      (make-instance 'HOLDING-ARC
                     :dist-to-td 100.0
                     :label "Kiowa holding point"
                     :symbol 'HOLD-KIOWA
                     ))

(setq HOLD-BYSON
      (make-instance 'HOLDING-ARC
                     :dist-to-td 100.0
                     :label "Byson holding point"
                     :symbol 'HOLD-BYSON
                     ))


(setq APP-DRAKO
      (make-instance 'APPROACH-ARC
                     :dist-to-td 100.0
                     :label "Drako approach path"
                     :symbol 'APP-DRAKO
```

```
                        ))

(setq APP-KEANN
      (make-instance 'APPROACH-ARC
                        :dist-to-td 100.0
                        :label "Keann approach path"
                        :symbol 'APP-KEANN
                        ))

(setq APP-COS
      (make-instance 'APPROACH-ARC
                        :dist-to-td 100.0
                        :label "COS approach path"
                        :symbol 'APP-COS
                        ))

(setq APP-KIOWA
      (make-instance 'APPROACH-ARC
                        :dist-to-td 100.0
                        :label "Kiowa approach path"
                        :symbol 'APP-KIOWA
                        ))

(setq APP-BYSON
      (make-instance 'APPROACH-ARC
                        :dist-to-td 100.0
                        :label "Byson approach path"
                        :symbol 'APP-BYSON
                        ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Interconnect Arcs in the Terminal Graph
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(send MISSED
      :set-next-arcs (list FF-DRAKO HOLD-DRAKO))

(send TD-ARC
      :set-next-arcs nil)

(send COR-DRAKO
      :set-next-arcs (list TD-ARC))

(send COR-KEANN
      :set-next-arcs (list TD-ARC))

(send COR-COS
      :set-next-arcs (list TD-ARC))

(send COR-KIOWA
      :set-next-arcs (list TD-ARC))

(send COR-BYSON
      :set-next-arcs (list TD-ARC))

(send FF-DRAKO
      :set-next-arcs (list COR-DRAKO))

(send FF-KEANN
      :set-next-arcs (list COR-KEANN))

(send FF-COS
      :set-next-arcs (list COR-COS))

(send FF-KIOWA
      :set-next-arcs (list COR-KIOWA))

(send FF-BYSON
      :set-next-arcs (list COR-BYSON))

(send HOLD-DRAKO
      :set-next-arcs (list FF-DRAKO))

(send HOLD-KEANN
      :set-next-arcs (list FF-KEANN))

(send HOLD-COS
      :set-next-arcs (list FF-COS))

(send HOLD-KIOWA
      :set-next-arcs (list FF-KIOWA))

(send HOLD-BYSON
      :set-next-arcs (list FF-BYSON))

(send APP-DRAKO
      :set-next-arcs (list FF-DRAKO HOLD-DRAKO))

(send APP-KEANN
      :set-next-arcs (list FF-KEANN HOLD-KEANN))

(send APP-COS
      :set-next-arcs (list FF-COS HOLD-COS))

(send APP-KIOWA
      :set-next-arcs (list FF-KIOWA HOLD-KIOWA))

(send APP-BYSON
      :set-next-arcs (list FF-BYSON HOLD-BYSON))
```

```
;;; -*- Mode: LISP; Package: MRS; Syntax: Zetalisp; Base: 10 -*-
;;;
;;; NB: Change this first line and all life as you know it will cease to exist...
;;;
;;;
;;;                        John Scoggins
;;;                  NASA/Ames Research Center
;;;                          M/S 210-9
;;;                   Moffett Field, CA  94035
;;;                        (415) 694-5431
;;;
;;; Overview
;;;
;;;  This file defines the flavors and methods used in the object base of
;;; the ATC Scheduler expert system.  For documentation about the syntax and
;;; features of the Flavors package, please see the Symbolics manual called
;;; "Reference Guide to Symbolics-Lisp", p. 417.  Since much of this file
;;; concerns the graphics and user interface packages, it is also helpful to
;;; see the Symbolics manual called "Programming the User Interface", p. 73.
;;;
;;;  This file will:
;;;
;;;     • Define the global variables for the system
;;;     • Define the flavors and methods for windows, routes and planes
;;;
;;;  All of this code is defined in terms of lisp functions or calls to the
;;; Flavors package, so it might be useful to compile this file for faster
;;; execution.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Define Global Variables
;;;
;;;  We use the function (DECLARE (SPECIAL <var>)) to define global variables
;;; instead of the typical use of SETQ.  This defines variables in terms
;;; of the Flavors package.  Primarily, this results in cleaner programming and
;;; less compiler warnings, but no change in the execution semantics.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(comment (Declare
  (special FRAME
           MENU-FLIGHT
           MENU-CLASS
           MENU-ROUTE
           MENU-TIME1
           MENU-TIME2
           MENU-ALT
           MENU-CAS
           MENU-ARC
           MENU-MORE
           MENU-FLIGHTS
           MENU-TRACE
           MENU-STEP
           MENU-GC
           ))
)

; Menu variable initial definitions

(setq MENU-ID 'PA001)
(setq MENU-FLIGHT nil)
(setq MENU-CLASS 'HEAVY)
(setq MENU-ROUTE 'DRAKO)
(setq MENU-TIME1 (time:get-universal-time))
(setq MENU-TIME2 (time:get-universal-time))
(setq MENU-ALT 36000.0)
(setq MENU-CAS 180.0)
(setq MENU-ARC 'TD-ARC)
(setq MENU-MORE nil)
(setq MENU-FLIGHTS nil)
(setq MENU-TRACE nil)
(setq MENU-STEP t)
(setq MENU-MINSTEP 4)
(setq MENU-SAVE 'SAVE)
(setq MENU-FILE "CHA:>Scoggins>TestCases.save")
(setq MENU-GC 'gc-off)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; PANE Flavor Definitions and Methods
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(DefFlavor LISTENER-PANE ((who-line  "")
                          )
           (tv:window)

  :gettable-instance-variables                    ;these flags mean that any of the
  :settable-instance-variables                    ;flavor's slot can be initialized,
  :initable-instance-variables                    ;accessed or reset
  )


(DefMethod (LISTENER-PANE :who-line-documentation-string)
           ()                                     ;explain mouse features

  who-line
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; FRAME Flavor Definitions and Methods
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(DefFlavor FRAME ((advisories nil)
                  (command-menu nil)
                  (listener nil)
                  (time-routes nil)
                  (min-time-step 4)
                  (last-time 0)
                  (time 0)              ;displayed time
                  (routes nil)
                  )
           (tv:bordered-constraint-frame-with-shared-io-buffer
            tv:basic-frame
            )
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables
  )


(DefMethod (FRAME :after :init)
           (ignore)

  (setq advisories   (send self :get-pane 'ADVISORIES)
        command-menu (send self :get-pane 'COMMAND-MENU)
        listener     (send self :get-pane 'LISTENER)
        time-routes  (send self :get-pane 'TIME-ROUTES)
        time         (time:get-universal-time)
        last-time    (time:get-universal-time)
        )
  (send self :set-save-bits t)                ;save screen when deexposed
  (send self :select-pane listener)
  )


(DefMethod (FRAME :command-loop)
           ()                                 ;command loop for system - this enables
                                              ;the use of mouse/menu driven user input
  (loop do (catch-error-restart
             ((ERROR SYS:ABORT) "Return to the ATC Scheduler prompt")

             (send self :expose)
             (setq TERMINAL-IO listener)
             (send listener :fresh-line)

             (cond ((or MENU-STEP
                        (send listener :listen)
                        )
                    (multiple-value-bind (value flag)
                        (with-input-editing-options
                            ((:preemptable :blip)
                             (:prompt "ATC Scheduler: ")
                             )
                          (read-or-end listener 'read-command-or-form))
                      (selectq flag
                        (:end
                         (format t "~&~A" "Leaving the ATC Scheduler...")
                         (return t)
                         )
                        (:blip
                         (selectq (car value)
                           (:menu
                            (catch 'MENU-ABORT
                              (format listener "~&~A" (eval (caddadr value)))
                              )
                            )
                           (:mouse-button
                            (selectq (cadr value)
                              (#\mouse-1-1
                               nil
                               )
                          (#\mouse-m-1
                           (setq MENU-STEP (not MENU-STEP))
                           (if (not MENU-STEP)
                               (send listener :set-more-p nil)
                               )
                           )
                          (#\mouse-r-1
                           nil
                           )
                          )
                        )
                       (otherwise
                        (format listener "Random blip -- ~S" value)))
                     )
                    (:command
                     (format listener "Execute ~:C command" (second value))
                     )
                    (otherwise
                     (send listener :fresh-line)
                     (format listener "~&~A" (eval value))
                     )
                    )
                  )
                )
               ((cond ((and (not MENU-STEP)
                            (< min-time-step
                               (abs (- last-time
                                       (time:get-universal-time))))
                            )
                       (send FRAME :update-time)
                       (send FRAME :draw-time-routes)
                       (send TERMINAL-GRAPH :update time))
                       ))
               )
           )
          )
  )


(DefMethod (FRAME :draw-time-routes)
           ()
  ;Draw the time lines
  (send time-routes :clear-window)
  (loop for (route . lines) in routes
        do (eval `(send ,time-routes :draw-lines tv:alu-ior ,@lines))
        do (send time-routes :draw-string (get-pname route) 0 (+ 12 (cadr lines)))
           )
  ;Draw each flight
  (loop for (flight arc enter exit dis alt cas)
            in (send TD-ARC :entering-flights)
        with max-x = (send time-routes :width)
        with scale = (// max-x 4000.0)
        with front = nil
        for x-pos = (- max-x (round (* (- enter time) scale)))
        for y-pos = (nth 2 (assoc (send flight :current-route) routes))
        for x-sep first 0 then (round (* scale (send flight :time-sep front)))
        do (send time-routes :draw-string (send flight :label) x-pos (- y-pos 12))
        do (send time-routes :draw-char fonts:ATCFONT (send flight :icon) x-pos y-pos)
        do (send time-routes :draw-rectangle x-sep 3 x-pos (1- y-pos))
        do (setq front flight)
           )
  ;Draw the current time
  (send time-routes                           ;redisplay the current time - the bit logic
        :draw-string                          ;function TV:ALU-XOR seems to right justify
        (format nil "Current Time: ~\time\" time)
        (- (send time-routes :width) 100)
        (- (send time-routes :height) 20)
        tv:alu-xor
        )
  )


(DefMethod (FRAME :print-advisories)
           (advs)
  ;Display a list of new advisories onto
      ADVISORIES pane
  (loop for m in (reverse
                    (cond ((atom advs) (list advs))
                            (t advs)))
        do (send advisories :home-cursor)
        do (send advisories :insert-line)
        do (send advisories :string-out m)
       )
  )


(DefMethod (FRAME :reset-time)
           (new-time)
  ;
  (setq time new-time
        last-time (time:get-universal-time)
        )
  )


(DefMethod (FRAME :update-time)
           ()
  ;
  (let* ((this-time (time:get-universal-time)))
    (setq time (+ time (- this-time last-time))
          last-time this-time
          )
    time
    )
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Command Menu Items
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(DefVar MENU-ITEMS
        '(("New Arrival"
           :value (let* ()
                    (setq MENU-TIME1 (send FRAME :update-time))
                    (tv:choose-variable-values
                      '((MENU-ID "Flight ID" :expression)
                        (MENU-CLASS "Class" :choose
                                    (HEAVY LARGE SMALL))
                        (MENU-ROUTE "Current Route" :choose
                                    (DRAKO KEANN COS KIOWA BYSON MISSED))
                        (MENU-TIME1 "Time @ Feeder Fix" :date)
                        (MENU-ALT "Current Altitude" :decimal-number)
                        (MENU-CAS "Current CAS" :decimal-number)
                        )
                      :label "Arrival Flight Parameters"
                      :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                        )
                      )
                    (PLAN-NEW-ARRIVAL
                      MENU-ID MENU-CLASS MENU-ROUTE MENU-TIME1 MENU-ALT MENU-CAS))
           :documentation " Create a new arrival"
           )

          ("Blocked Runway"
           :value (let* ()
                    (setq MENU-TIME1 (send FRAME :update-time))
                    (setq MENU-TIME2 MENU-TIME1)
                    (tv:choose-variable-values
                      '((MENU-TIME1 "From " :date)
                        (MENU-TIME2 "Until" :date)
                        )
                      :label "Blocked Runway Parameters"
                      :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                        )
                      )
                    (stash '(blocked-runway ,MENU-TIME1 ,MENU-TIME2))
                    (send TD-ARC :block-runway MENU-TIME1 MENU-TIME2)
                    )
           :documentation " Signal a blocked runway condition"
           )

          ("Missed Approach"
           :value (let* ()
                    (setq MENU-FLIGHT nil)
                    (tv:choose-variable-values
                      '((MENU-FLIGHT "Flight to miss approach"
                                     :assoc ,(mapcar '(lambda (flt)
                                                        (cons (send flt :label) flt))
                                                     (send TERMINAL-GRAPH :flight-list)
                                                     ))
                        )
                      :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                        )
                      )
                    (if MENU-FLIGHT
                        (PLAN-MISSED-APPROACH MENU-FLIGHT))
                    )
           :documentation " Signal a missed approach condition"
           )

          ("Display Flight Plan"
           :value (let* ((old-more-p (send TERMINAL-IO :more-p))
                         )
                    (send TERMINAL-IO :set-more-p t)
                    (setq MENU-FLIGHT nil)
                    (tv:choose-variable-values
                      '((MENU-FLIGHT "Flight to display"
                                     :assoc ,(mapcar '(lambda (flt)
                                                        (cons (send flt :label) flt))
```

```
                                                (send TERMINAL-GRAPH :flight-list)
                                                ))
                   )
                  :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                    )
                  )
                (send TERMINAL-IO :clear-window)
                (if MENU-FLIGHT
                    (send MENU-FLIGHT :report-plan))
                (send TERMINAL-IO :set-more-p old-more-p)
                )
      :documentation " Display a report of some flight's scheduled plan"
      )

     ("Obliterate Flight"
      :value (let* ()
                (setq MENU-FLIGHT nil)
                (tv:choose-variable-values
                  '((MENU-FLIGHT "Flight to remove"
                                 :assoc ,(mapcar '(lambda (flt)
                                                     (cons (send flt :label) flt))
                                                (send TERMINAL-GRAPH :flight-list)
                                                ))
                   )
                  :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                    )
                  )
                (if MENU-FLIGHT
                    (send MENU-FLIGHT :remove-self))
                )
      :documentation " Disavow any information concerning a particular flight"
      )

     ("Modify Clock"
      :value (let* ()
                (setq MENU-TIME1 (send FRAME :update-time))
                (tv:choose-variable-values
                  '((MENU-TIME1 "New Time" :date)
                    (MENU-MINSTEP "Minimum Time Step" :integer)
                    (MENU-STEP "Time Stepping Mode" :assoc (("Real-Time" . nil)
                                                            ("Manual-Step" . t)
                                                            ))
                    )
                  :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                    )
                  )
                (if (not MENU-STEP)
                    (send TERMINAL-IO :set-more-p nil)
                    )
                (send FRAME :reset-time MENU-TIME1)
                )
      :documentation " Change the system clock parameters"
      )

     ("Save or Restore"
      :value (let* ()
                (tv:choose-variable-values
                  '((MENU-FILE "Scenario file" :pathname)
                    (MENU-SAVE "Perform with file" :choose (SAVE RESTORE))
                    )
                  :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                    )
                  )
                (selectq MENU-SAVE
                  (SAVE
                   (send TERMINAL-GRAPH :save-scenario MENU-FILE))
                  (RESTORE
                   (send TERMINAL-GRAPH :restore-scenario MENU-FILE))
                  )
                )
      :documentation " Use a saved scenario file to control the simulation"
      )

     ("More Processing"
      :value (let* ()
                (tv:choose-variable-values
              '((MENU-MORE "Listener **MORE** processing" :assoc (("On" . t)
                                                                  ("Off" . nil)
                                                                  ))
               )
              :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                )
              )
            (if MENU-MORE
                (setq MENU-STEP t)
                )
            (send TERMINAL-IO :set-more-p MENU-MORE)
            )
     :documentation " Toggle Listener **MORE** processing"
     )

    ("MRS Task Tracing"
     :value (let* ()
            (tv:choose-variable-values
              '((MENU-TRACE "MRS task tracing" :assoc (("On" . t)
                                                       ("Off" . nil)
                                                       ))
               )
              :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                )
              )
            (if MENU-TRACE
                (tracetask '&x)
                (untracetask)
                )
            )
     :documentation " Toggle MRS task tracing"
     )

    ("Garbage Collection"
     :value (let* ((old-state MENU-GC)
                   )
            (tv:choose-variable-values
              '((MENU-GC "Garbage collection" :assoc (("On" . gc-on)
                                                      ("Off" . gc-off)
                                                      ("Immediately" . gc-immediately)
                                                      ("Status" . gc-status)
                                                      ))
               )
              :margin-choices '(("Abort" (throw 'MENU-ABORT "Menu Options Aborted"))
                                )
              )
            (apply MENU-GC nil)
            (if (eq MENU-GC 'gc-status)
                (setq MENU-GC old-state)
                )
            )
     :documentation " Change the garbage collection state"
     )
    )
 )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Compile, Load Flavors; and Instantiate Objects
;;;
;;;  We give the compiler the names of each flavor to be used in the system.
;;; Then as this file is compiled, all flavor definitions and methods will be
;;; compiled into the bin file.  Also, data structures for all the flavors will
;;; then be created at load time.  Otherwise the flavors and methods would be
;;; compiled at run time.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(Compile-Flavor-Methods
  LISTENER-PANE
  FRAME
  )


(setq FRAME
      (tv:make-window 'FRAME
                      :routes
                      '((Drako  0 50 900 50 900 100 1067 100)
                        (Keann  0 100 1067 100)
                        (COS    0 150 900 150 900 100 1067 100)
                        (Kiowa  0 200 900 200 900 100 1067 100)
                        (Byson  0 250 900 250 900 100 1067 100)
                        (Missed 0 300 900 300 900 100 1067 100)
                        )

                      :panes
                      '((COMMAND-MENU tv:command-menu-pane
                                      :item-list ,MENU-ITEMS
                                      )

                        (TIME-ROUTES tv:window-pane
                                     :font-map ,(list fonts:h17)
                                     :label "Time Routes"
                                     :blinker-p nil
                                     )

                        (ADVISORIES tv:window-pane
                                    :font-map ,(list fonts:tvfont)
                                    :label "Advisories"
                                    :more-p nil
                                    :blinker-p nil
                                    )

                        (LISTENER LISTENER-PANE
                                  :font-map ,(list fonts:tvfont)
                                  :label "Listener"
                                  :more-p nil
                                  :who-line
                                  " M: Real Time on//off..."
                                  )
                        )

                      :constraints
                      '((main . ((TIME-ROUTES bottom-strip)
                                 ((bottom-strip :horizontal (0.5)
                                                (ADVISORIES COMMAND-MENU LISTENER)
                                                ((ADVISORIES   0.46)
                                                 (COMMAND-MENU 0.14)
                                                 (LISTENER     0.40))))
                                 ((TIME-ROUTES :even))))
                        )
                      )
      )
```

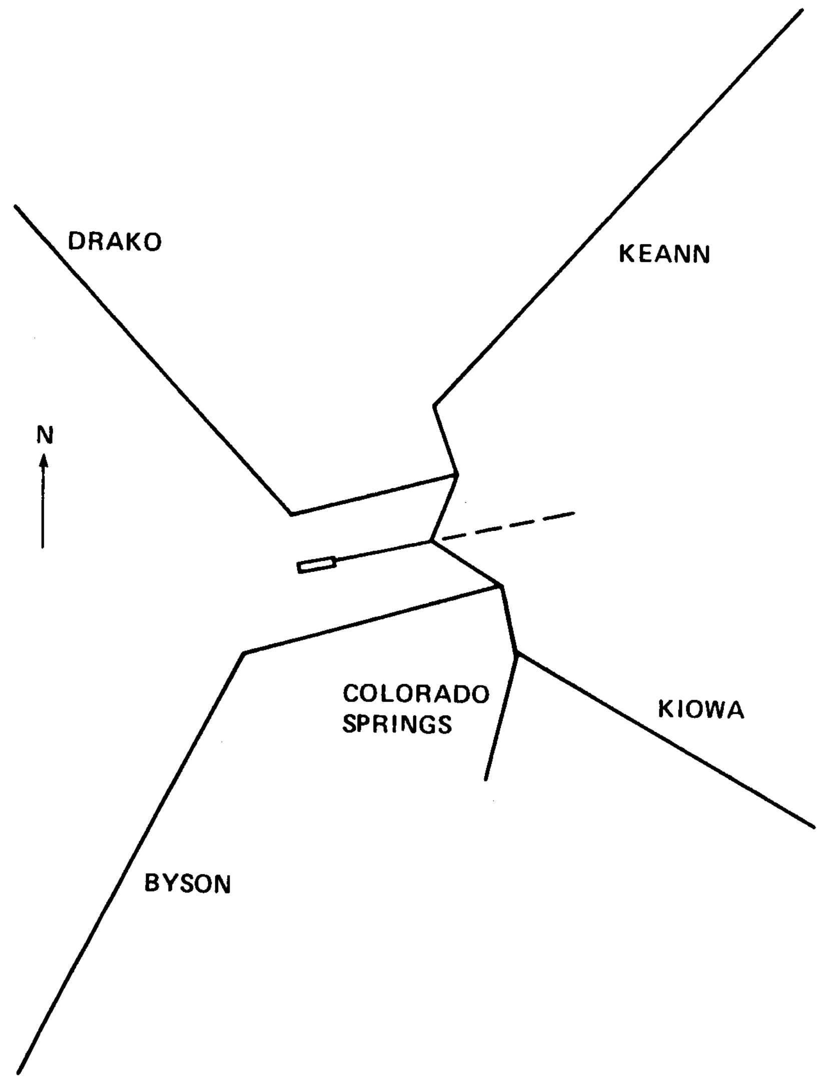


Fig. 1 Denver Route Structure.

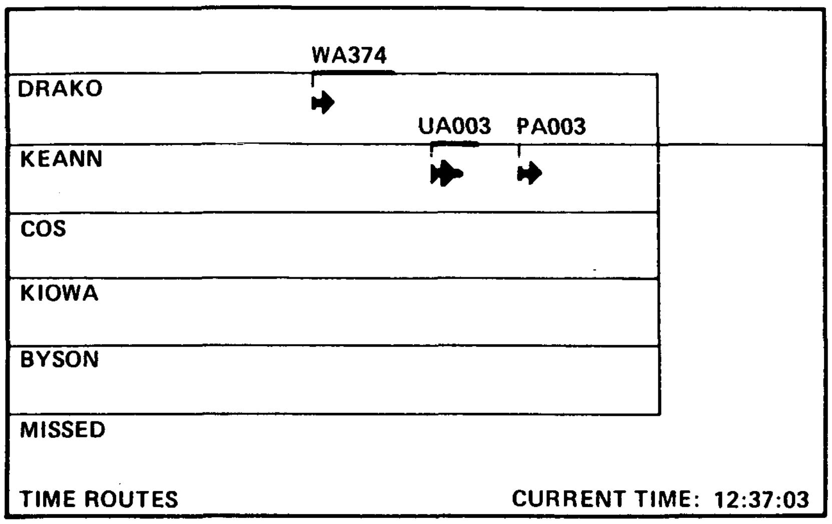


Fig. 2 Schedule for New Arrival.

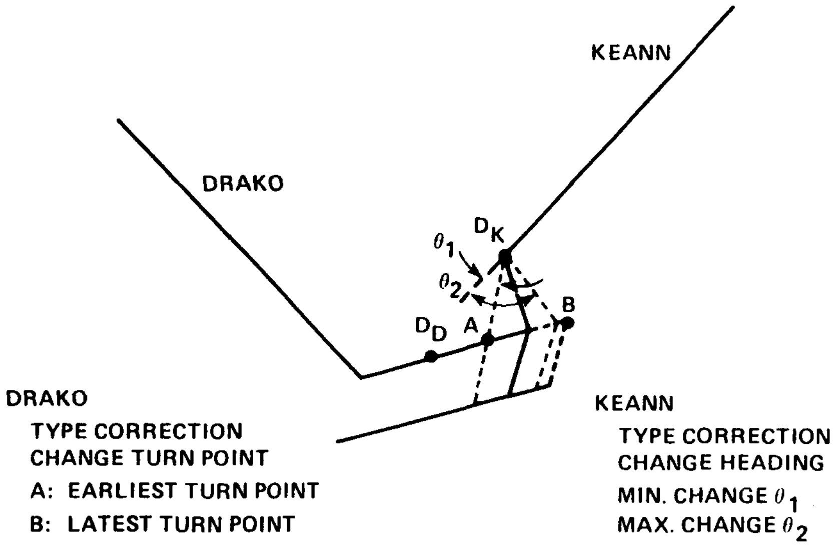


Fig. 3 Path Correction.

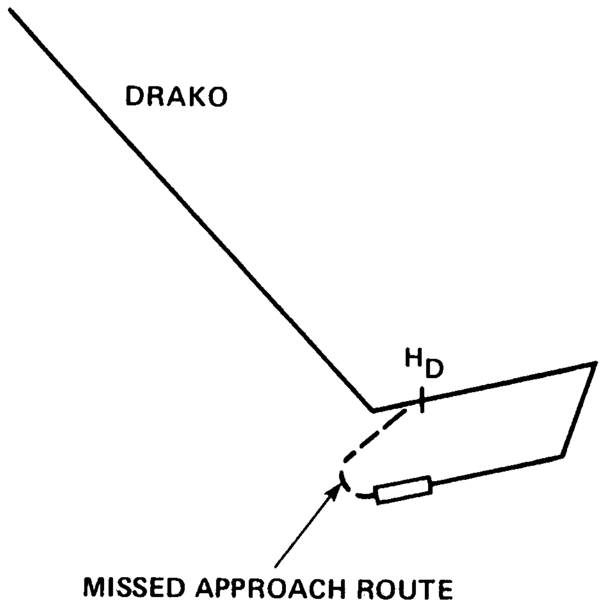
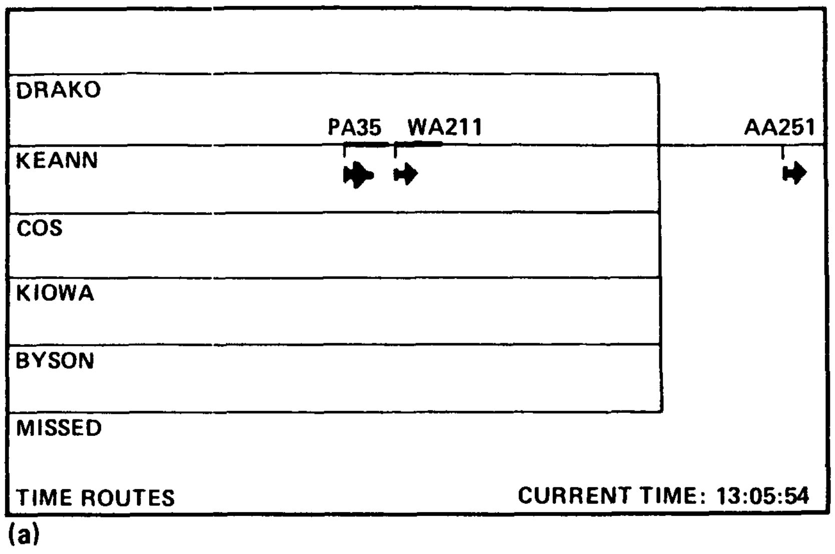


Fig. 4 Missed Approach Route.



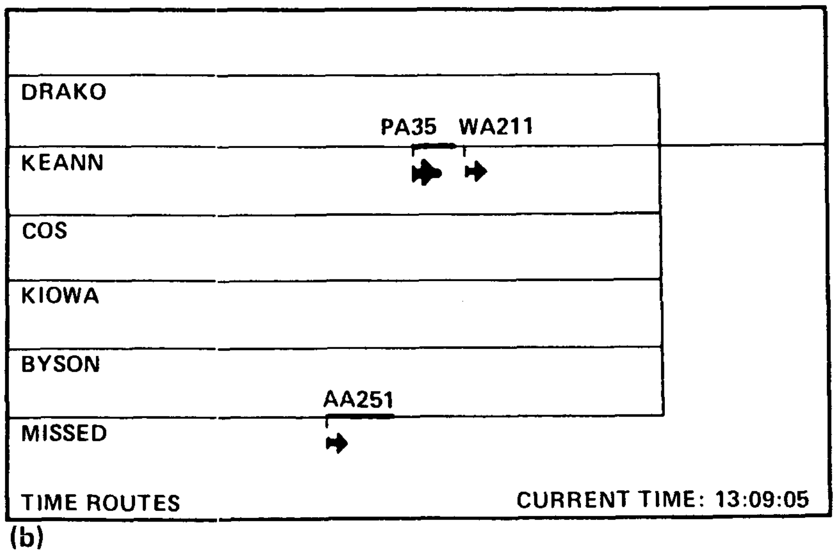
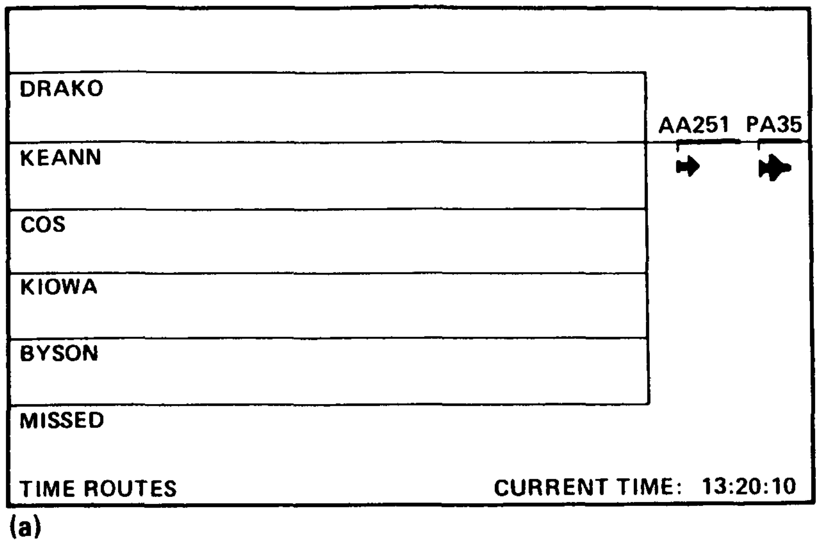
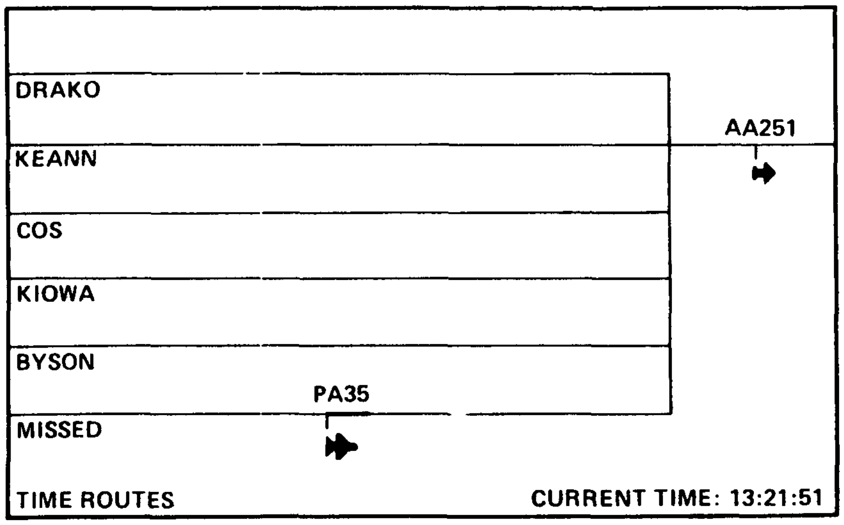


Fig. 5 Time Line. (a) Just before Missed Approach for AA251. (b) Missed Approach for AA251.

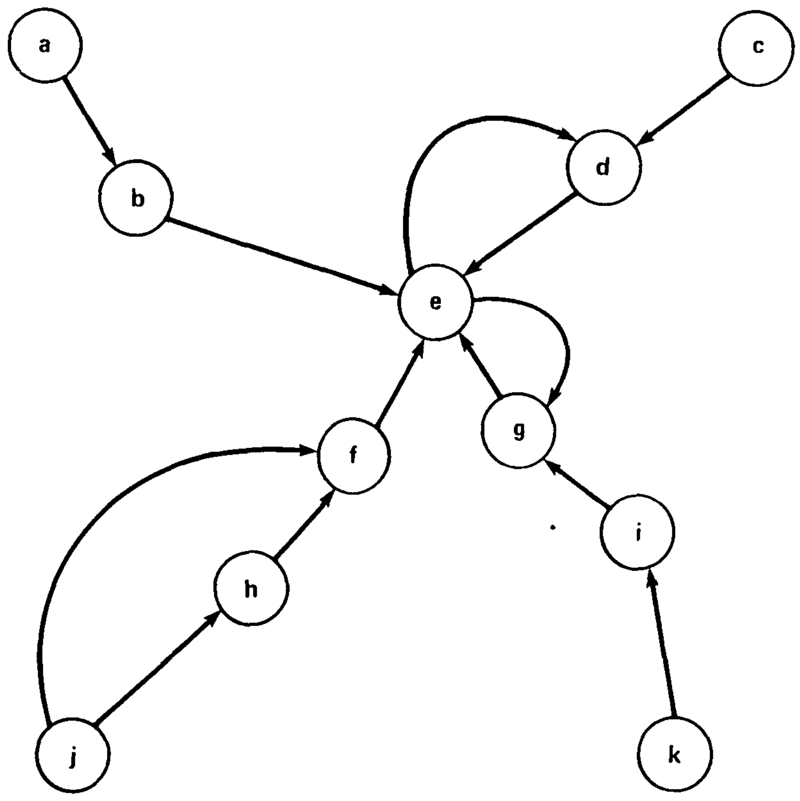


(a)



(b)

Fig. 6 Time Line. (a) Runway Closure. (b) Revised Schedule Due to Runway Closure.

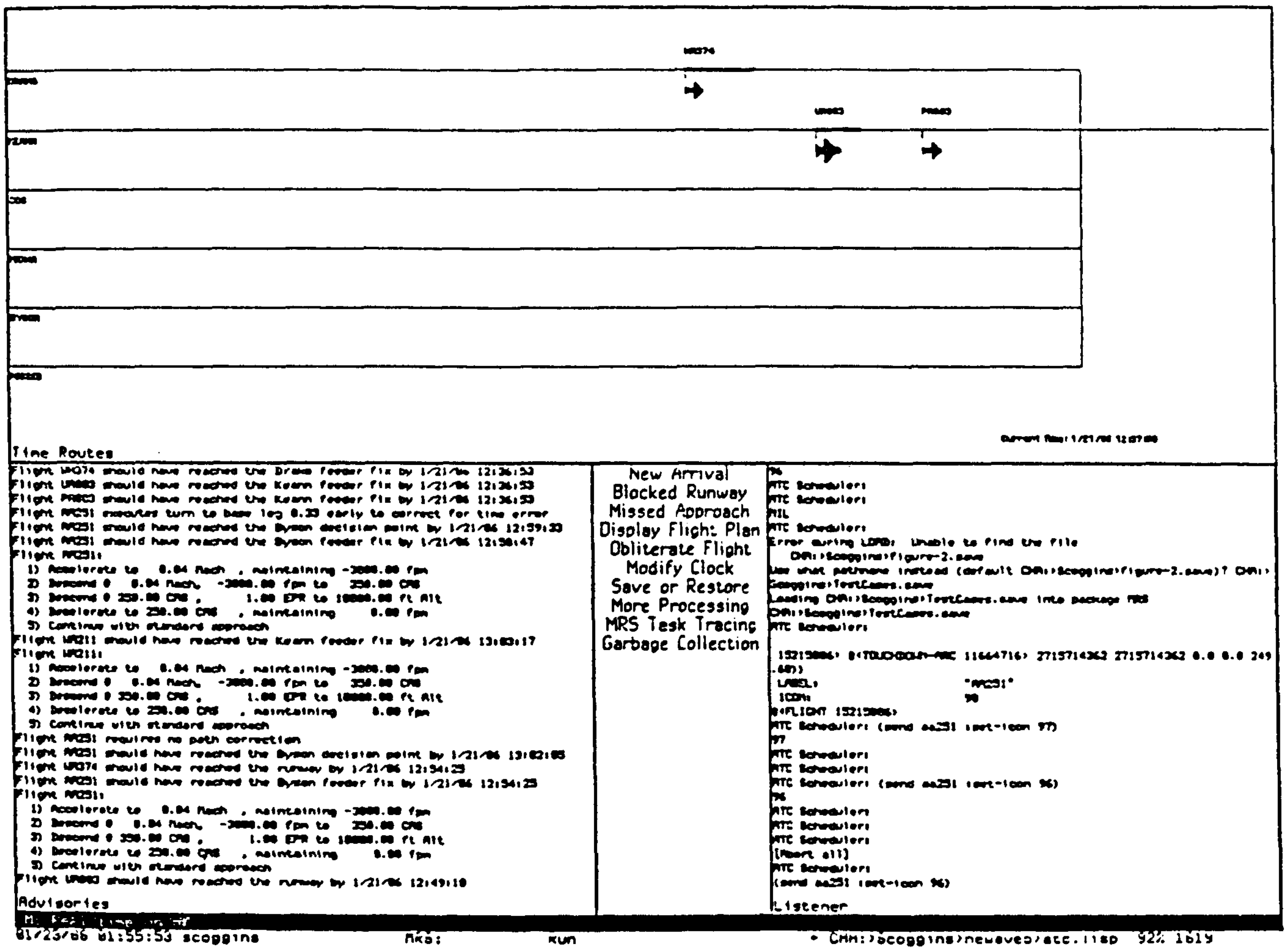


INFORMATION FOR ARC J → H

DISTANCE-TO-TOUCHDOWN:	165 n.mi.
ALTERNATE-PATH:	J → F
ENTRY-CODE:	FEEDER FIX ENTRY PROC
EXIT-CODE:	FEEDER FIX EXIT PROC
ENTERING-FLIGHTS:	(PA35 13:02:23 13:11:45 36000 182)
	(VA07 13:10:55 13:20:01 34500 178)
EXITING-FLIGHTS:	(AA24 12:50:33 12:59:57 32800 191)
	(XX01 12:55:41 13:03:28 29000 193)

Fig. 7 Terminal area represented as a direct graph.

ORIGINAL PAGE IS
OF POOR QUALITY



```
Current Time: 1/21/86 12:07:00
```

```
Flight WA074 should have reached the Drake feeder fix by 1/21/86 12:36:53
Flight UA803 should have reached the Kearn feeder fix by 1/21/86 12:36:53
Flight PA803 should have reached the Kearn feeder fix by 1/21/86 12:36:53
Flight AA251 executes turn to base leg 0.33 early to correct for time error
Flight AA251 should have reached the Byson decision point by 1/21/86 12:59:33
Flight AA251 should have reached the Byson feeder fix by 1/21/86 12:58:47
Flight AA251:
  1) Accelerate to   0.84 Mach  , maintaining -3000.00 fpm
  2) Descend @   0.84 Mach,  -3000.00 fpm to   250.00 CAS
  3) Descend @ 250.00 CAS ,      1.00 EPR to 10000.00 ft Alt
  4) Decelerate to 250.00 CAS   , maintaining     0.00 fpm
  5) Continue with standard approach
Flight WA211 should have reached the Kearn feeder fix by 1/21/86 13:03:17
Flight WA211:
  1) Accelerate to   0.84 Mach  , maintaining -3000.00 fpm
  2) Descend @   0.84 Mach,  -3000.00 fpm to   350.00 CAS
  3) Descend @ 350.00 CAS ,      1.00 EPR to 10000.00 ft Alt
  4) Decelerate to 250.00 CAS   , maintaining     0.00 fpm
  5) Continue with standard approach
Flight AA251 requires no path correction
Flight AA251 should have reached the Byson decision point by 1/21/86 13:02:05
Flight WA074 should have reached the runway by 1/21/86 12:54:25
Flight AA251 should have reached the Byson feeder fix by 1/21/86 12:54:25
Flight AA251:
  1) Accelerate to   0.84 Mach  , maintaining -3000.00 fpm
  2) Descend @   0.84 Mach,  -3000.00 fpm to   250.00 CAS
  3) Descend @ 350.00 CAS ,      1.00 EPR to 10000.00 ft Alt
  4) Decelerate to 250.00 CAS   , maintaining     0.00 fpm
  5) Continue with standard approach
Flight UA803 should have reached the runway by 1/21/86 12:49:10
```

```
96
ATC Scheduler:
ATC Scheduler:
NIL
ATC Scheduler:
Error during LOAD:  Unable to find the file
   CHA:>Scoggins>figure-2.save
Use what pathname instead (default CHA:>Scoggins>figure-2.save)? CHA:>
Scoggins>TestCases.save
Loading CHA:>Scoggins>TestCases.save into package MRS
CHA:>Scoggins>TestCases.save
ATC Scheduler:

 15215806> #<TOUCHDOWN-ARC 11664716> 2715714362 2715714362 0.0 0.0 249
.60))
 LABEL:                         "AA251"
 ICON:                          98
#<FLIGHT 15215806>
ATC Scheduler: (send aa251 :set-icon 97)
97
ATC Scheduler:
ATC Scheduler:
ATC Scheduler: (send aa251 :set-icon 96)
96
ATC Scheduler:
ATC Scheduler:
ATC Scheduler:
[Abort all]
ATC Scheduler:
(send aa251 :set-icon 96)
```

```
01/23/86 01:55:53 scoggins          Mxs:          Run          • CHA:>scoggins>newaves>atc.lisp  92% 1619
```

Fig. 8 User interface.

1. Report No. NASA TM-88234	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle TIME-BASED AIR TRAFFIC MANAGEMENT USING EXPERT SYSTEMS		5. Report Date April 1986
		6. Performing Organization Code
7. Author(s) L. Tobias and J. L. Scoggins		8. Performing Organization Report No. A-86188
9. Performing Organization Name and Address Ames Research Center Moffett Field, CA 94035		10. Work Unit No.
		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Technical Memorandum
		14. Sponsoring Agency Code 505-66-1

15. Supplementary Notes

Point of Contact: L. Tobias, Ames Research Center, M/S 210-9, Moffett Field, CA 94035. (415) 694-5430 or FTS 464-5430

16. Abstract

A prototype expert system has been developed for the time scheduling of aircraft into the terminal area. The three functions of the air-traffic-control schedule advisor are as follows: First, for each new arrival, it develops an admissible flight plan for that aircraft. Second, as the aircraft progresses through the terminal area, it monitors deviations from the aircraft's flight plan and provides advisories to return the aircraft to its assigned schedule. Third, if major disruptions such as missed approaches occur, it develops a revised plan. The advisor is operational on a Symbolics 3600, and is programmed in MRS (a logic programming language), Lisp, and Fortran.

17. Key Words (Suggested by Author(s)) Air traffic control Expert system Scheduling		18. Distribution Statement Unlimited Subject Category - 03	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 58	22. Price* A04

*For sale by the National Technical Information Service, Springfield, Virginia 22161