A RAND NOTE

AUTOPILOT: A DISTRIBUTED PLANNER FOR AIR FLEET CONTROL

Perry Thorndyke, David McArthur,
Stephanie Cammarata

July 1981

N-1731-ARPA

Prepared For The Defense Advanced Research Projects Agency

Rand
SANTA MONICA, CA. 90406

The research described in this report was sponsored by the Defense Advanced Research Projects Agency under ARPA Order No. 3460/3473, Contract No. MDA903-78-C-0029, Information Processing Techniques

Published by The Rand Corporation

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AN 1/

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER N-1731-ARPA	2. GOVT ACCESSION NO. AD-A107139	3. RECIPIENT'S CATALOG NUMBER
4. ~~TITLE (and Subtitle)~~ 6 AUTOPILOT: A Distributed Planner for Air Fleet Control.		5. TYPE OF REPORT & PERIOD COVERED 9 Interim rept.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 Perry W. Thorndyke, David McArthur, Stephanie Cammarata		8. CONTRACT OR GRANT NUMBER(s) 15 MDA903-78-C-0029
9. PERFORMING ORGANIZATION NAME AND ADDRESS The Rand Corporation 1700 Main Street Santa Monica, CA. 90406		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 11
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Department of Defense Arlington, VA 22209		12. ~~REPORT DATE~~ Jul[illegible]81
		13. NUMBER OF PAGES 25
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 34		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release: Distribution Unlimited

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

No Restrictions

18. SUPPLEMENTARY NOTES

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

Distribution Systems	Air Traffic Control
Positioning	Flight Paths
Routing	Heuristic Methods

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

See Reverse Side

216600

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Distributed planning requires both architectures for structuring multiple planners and techniques for planning, communication, and cooperation. We describe a family of systems for distributed control of multiple aircraft, in which each aircraft plans its own flight path and avoids collisions with other aircraft. AUTOPILOT, the kernel planner used by each aircraft, comprises several processing "experts" that share a common world model. These experts sense the world, plan and evaluate flight paths, communicate with other aircraft, and control plan execution. We discuss four architectures for the distribution of airspace management and planning responsibility among the several aircraft occupying the airspace at any point in time. The architectures differ in the extent of cooperation and communication among aircraft.

A RAND NOTE

AUTOPILOT: A DISTRIBUTED PLANNER FOR AIR FLEET CONTROL

Perry Thorndyke, David McArthur,
Stephanie Cammarata

July 1981

N-1731-ARPA

Prepared For The Defense Advanced Research Projects Agency

Rand
SANTA MONICA, CA. 90406

PREFACE

This Note summarizes the results of an initial attempt to design and implement distributed computer systems for planning and control. The research has focused on the design of architectures for the distribution of functions, the specification of kernel capabilities for each node in the distributed system, and the empirical evaluation of alternative distribution schemes. The results should interest researchers and system designers interested in techniques for multi-processor cooperation.

The research was sponsored by the Defense Advanced Research Projects Agency under Contract No. MDA903-78-C-0029. A shortened version of this Note appears in the Proceedings of the International Joint Conference on Artificial Intelligence, held in Vancouver, August 1981.

SUMMARY

Distributed planning requires both architectures for structuring multiple planners and techniques for planning, communication, and cooperation. We describe a family of systems for distributed control of multiple aircraft, in which each aircraft plans its own flight path and avoids collisions with other aircraft. AUTOPILOT, the kernel planner used by each aircraft, comprises several processing "experts" that share a common world model. These experts sense the world, plan and evaluate flight paths, communicate with other aircraft, and control plan execution. We discuss four architectures for the distribution of airspace management and planning responsibility among the several aircraft occupying the airspace at any point in time. The architectures differ in the extent of cooperation and communication among aircraft.

ACKNOWLEDGMENTS

This work has benefited from the substantial contributions of James Gillogly, Frederick Hayes-Roth, Randall Steeb, and Robert Wesson.

CONTENTS

I. INTRODUCTION

Distri .ed planning refers to the process by which multiple processors cooperate to achieve a set of common objectives. Development of distributed planning systems requires two major activities: the specification of architectures for structuring the set of cooperating processors, and the discovery and implementation of planning techniques to be used by each processor. We have undertaken both sets of activities in an effort to develop methods for distributed control of simulated aircraft moving through an air traffic control (ATC) sector. Adopting this task domain has permitted us to investigate four important questions concerning cooperation:

- What formalisms are required to represent individual processors and the interactions among them?
- What are the computational costs and benefits of different architectures for distributing planning functions?
- How should distributed planners cope with incomplete and erroneous information? Distributed planners typically possess different knowledge bases, and no individual has a complete and accurate world model. Such differences increase the complexity of coordinating planning efforts.
- What are the pragmatics of cooperation? Distributed planning should be superior to centralized planning only if methods can be devised for coordinating the activities of the multiple processors.

These methods must consider the tradeoffs between local planning and requests for cooperation, inferring intentions and requesting information from others, and synchrony and asynchrony among multiple processors working on different aspects of a common problem.

To investigate these questions, we have implemented a planner called AUTOPILOT. AUTOPILOT simulates the sensing, route planning, and communications activities of a single aircraft flying through a high-traffic air sector. It controls the aircraft by cooperating with virtual clones of itself, each of which is assigned to and controls a different aircraft. In this Note, we describe the planning techniques embodied in four specific versions of AUTOPILOT that differ in the amount of communication and cooperation among the multiple planners. By implementing alternative versions, we hope to (1) derive a set of cooperative planning methods that are robust across architectures and planning environments and (2) support machine experiments that evaluate the performance of different cooperation regimes.

The Note is organized as follows. We first describe briefly the ATC simulation that provides our task domain. We then discuss the kernel design of AUTOPILOT that is invariant across the different versions we have investigated. Next, we present the four object-centered planning architectures we have investigated to date and the changes in AUTOPILOT that these versions entail. Finally, we draw some tentative conclusions concerning the questions we have posed above.

II. THE ATC TASK DOMAIN

The task environment for AUTOPILOT is provided by a real-time ATC simulation. Figure 1 illustrates the airspace used by the simulation. The airspace includes airways (indicated by commas) that link entry/exit fixes (0-9) at the airspace boundaries, two airports (% and #), and two navigation aids (* and !) through which aircraft can be vectored. During each run of the simulation, 26 aircraft arrive in the airspace at random times. Every aircraft enters the airspace at a particular entry fix or originates its flight at one of the airports. Aircraft must be issued commands to depart, land, change course, and/or change altitude in order to successfully navigate them to their destinations. The simulation provides two types of information: the airspace display and the flight plans for active and approaching aircraft. The airspace display (shown on the left side of Fig. 1) portrays the locations of all aircraft in the airspace, their identifiers, and their altitudes in thousands of feet (e.g., A5, X6). Every 15 seconds the display is updated and the aircraft move 1 mile (to an adjacent "." or ",") in the direction of their current heading. The flight plan for each aircraft (shown on the right side of Figure 1) displays, reading left to right, its status (active or approaching), its identifier, its aircraft type (p=propeller, j=jet), its current location (or origin, for pending aircraft), its destination, its altitude (in thousands of feet), and its heading. For example, R, a propeller aircraft, will enter the airspace in one time-step of the simulation at infix location 8, heading northwest at an altitude of 6000 feet. Its destination is exit fix 0.

A potential route for R would take R northwest to navigation aid "!" and then north to 0.

Successful control of aircraft requires landing planes at their desired airports in prescribed descent patterns or sending them out of the airspace at the desired fix, with the correct heading (i.e., along the airway), and at an altitude of 5000 feet. All aircraft must always maintain at least 3 miles of horizontal separation or 1000 feet of vertical separation. A violation of any of these constraints produces an error, as does allowing an aircraft to exhaust its fuel supply.

```
        AIRSPACE DISPLAY                FLIGHT PLANS

. . . . 0 . . . . 6 . . . . .          Xj ,->9 6  SW
7 . . . , . . . X6. . . . . .          Aj ,->2 5  SE
. A5. . , . . , . . . . . . .        1 Rp 8->0 6  NW
. . , . , . , . . . . . . . .
. . . , , , . . . . . . . . .
. . . . * . . . . . . . . . .
. . . , , , . . . . . . . . .
. . , . , . , . . . . . . . 4
. , . . , . . , . . . . . , .
3 . . . , . . . , . . . , . .
. . . . , . . . . , . , . . .
. . . . % . . . . . # . . . .
. . . . , . . . . , . , . . .
1 . . . , . . . , . . . , . .
. , . . , . . , . . . . . , .
. . , . , . , . . . . . . . 2
. . . , , , . . . . . . . . .
. . . . ! . . . . . . . . . .
. . . , , , . . . . . . . . .
. . , . , . , . . . . . . . .
. , . . , . . , . . . . . . .
5 . . . , . . . , . . . . . .
. . . . , . . . . , . . . . .
. . . . 9 . . . . . 8 . . . .
```

Fig. 1—Output from the ATC simulation

III. THE DISTRIBUTION OF PLANNING EFFORT

In conventional, real-world ATC, a single controller plans and controls all aircraft in the airspace. In our simulation, planning responsibility is distributed among the aircraft themselves. Each aircraft is controlled by an automated planner called AUTOPILOT. We refer to this allocation of function as an object-centered architecture for distributed planning. Thus, whenever a new aircraft appears in the airspace, a new AUTOPILOT clone is created and performs all planning and cooperation for that aircraft as it navigates from its origin to its destination.

All AUTOPILOT clones are behaviorally identical and can be viewed as virtual copies of a generic ATC planner. The structure of AUTOPILOT most closely resembles that of an independent actor (Hewitt, 1977) or object, as in SMALLTALK (Kay, 1972), DIRECTOR (Kahn, 1978), or ROSS (McArthur & Sowizral, 1981). Specifically, AUTOPILOT has a repertoire of sensing, planning, evaluation, and execution behaviors that can be triggered by receipt of messages from other aircraft.

In the current implementation, we simulate multiple planners by a single planning system that assumes different perspectives for each aircraft. The planner spawns offspring for different aircraft that contain the data base and world model specific to each. The computational expertise resides in the generic planner and can be applied to any of the various data bases. Thus, as in object-oriented programming languages, behavior rules reside with the generic planner and can be inherited by individual objects. Object-specific world views and knowledge reside in data structures unique to each individual aircraft.

IV. THE AUTOPILOT DESIGN

AUTOPILOT contains a design kernel common to all the architectural variants we have investigated. Figure 2 illustrates this kernel.

Several processing modules function as experts that share data and results via a common data base, the world model. As in a Hearsay-like model (Erman et al., 1980; Hayes-Roth et al., 1979), performance of these experts is triggered by particular conditions in the world model, and each expert posts its results in the world model as new knowledge or changes to existing knowledge. The world model contains such information as aircraft locations, their flight plans or assumed flight plans, and its own destination and tentative plans.

The sensor receives simulated radar returns in the form of airspace displays every 15 seconds. By comparing new displays with knowledge in the world model, the sensor compiles a list of location updates to be posted.

When AUTOPILOT is assigned to a new aircraft, the plan generator produces a set of tentative flight plans to navigate the aircraft from its entry fix to its exit fix. The evaluator tests these tentative plans against the real or inferred flight plans of other aircraft and posts predicted conflicts in the world model. The plan generator uses this information either to attach a minor patch to an existing plan or to replan completely.

The communicator exchanges information and requests with other aircraft. When planning is thwarted by environmental uncertainty, the communicator may request locations or intentions from others. If the plan

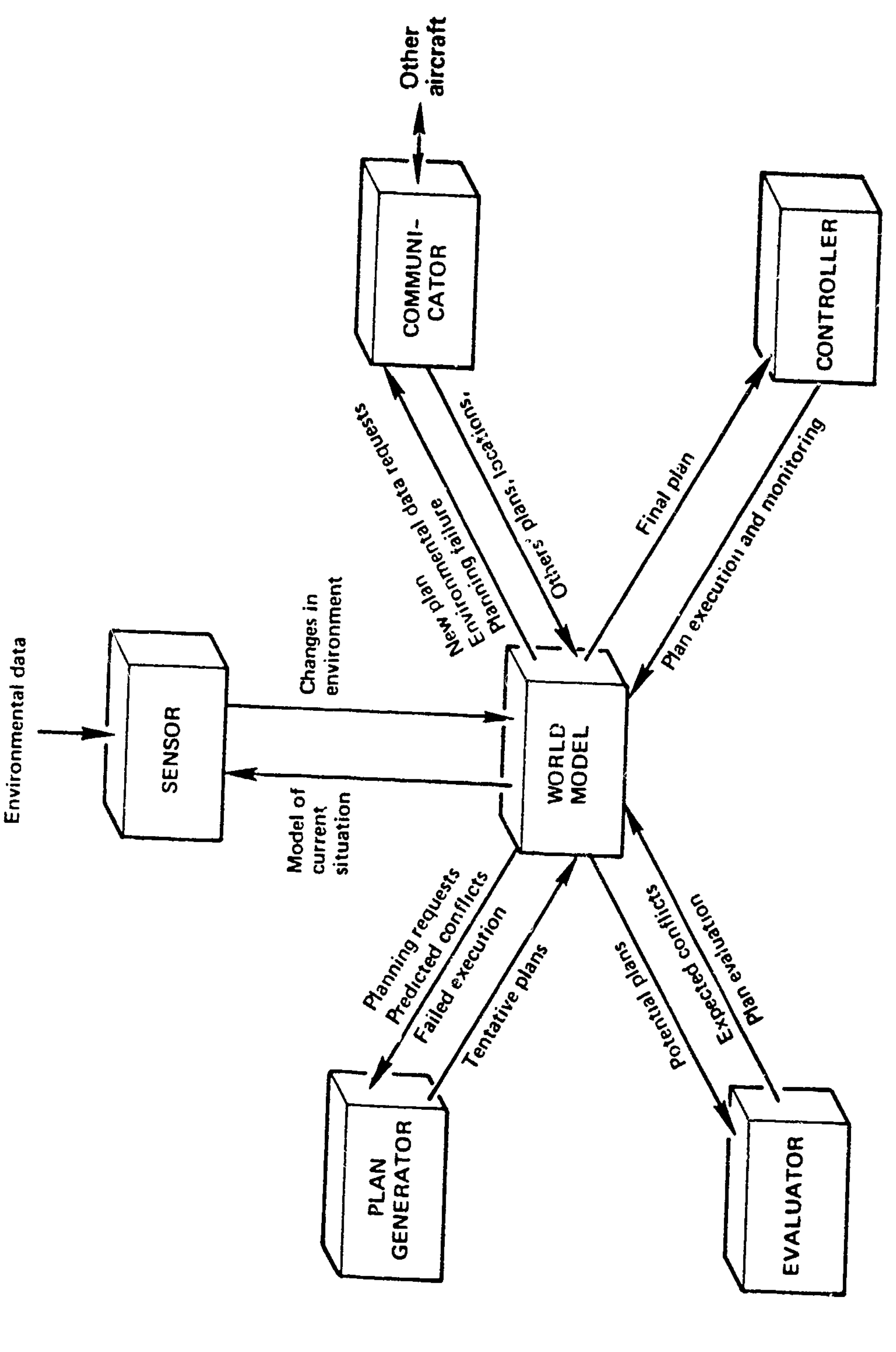

Fig. 2—Structure of the kernel planner

generator has failed to produce a conflict-free route, the communicator may request other aircraft to patch or replan their routes.

Finally, the controller implements the aircraft's flight plan. It monitors the location of the aircraft and issues commands to alter course or altitude at the appropriate locations in the airspace.

V. INCREMENTAL PLANNING

AUTOPILOT represents a plan in the world model as a schema with several slots to be filled during the planning process. For example, the completed plan for aircraft R in Fig. 1 is

```
(PLAN008
  AIRCRAFT      R
  COMMANDS      (a4 :0 a5)
  CONSTRAINTS   ((10 23) (10 23) (4 1))
  ROUTE         ((10 23 7 285) (9 22 6 300) ... (4 0 5 630))
  CONFLICTS     2
  CONFLICTSUM   ((X (4 17)) (X (4 16)))
  LENGTH        24
  VALUE         106
  PARENT        NIL
  OFFSPRING     NIL).
```

The slots in the plan schema contain information about the commands required to execute the plan, the x-y coordinates at which the commands must be executed, estimates of the overall utility of the plan, annotations of the plan's current bugs (i.e., predicted conflicts), and a four-dimensional map of the executed plan (i.e., a specification of the location of the aircraft at each point in time). Slots in the plan schema are filled at different times during planning. Some contributions are made during plan generation, some during evaluation, and other during plan patching. We discuss these processes in more detail below.

PLAN GENERATION

AUTOPILOT produces route plans incrementally by planning approximately. The plan generator first produces with minimal effort a few standard routes from the entry fix to the desired destination. These plans are then evaluated to determine the nature and location of expected conflicts. Finally, the best plans are refined, using a variety of techniques to produce local patches that avoid the conflict situations. This incremental approach to planning has four advantages: First, it emphasizes general adherence to designated airways and conventional routing strategies. Second, plan failures are simple to diagnose and describe; therefore, it is possible to patch accurately. Third, the incremental planning strategy reflects the approach used by real air traffic controllers and by expert humans performing in the ATC simulation. Fourth, this strategy is well suited to the distributed planning environment, since predicted conflicts identify sets of aircraft that must cooperate to solve their common problem.

The plan generator produces several initial plans by indexing a library of plan templates. Each template is list of commands, indexed by infix/outfix, that is guaranteed to take an aircraft to its destination from its entry fix. Each infix/outfix pair has several library entries, denoting standard routes across the airspace. One entry corresponds to the shortest route along the designated airways. Other entries are produced by the application of heuristics that enforce 3-mile horizontal separation from the standard route across as much of the airspace as possible.

PLAN EVALUATION AND CONFLICT DETECTION

The evaluator detects conflicts in candidate plans, using a fast-time lookahead. Once candidate initial plans are generated, the evaluator computes a four-dimensional route map of the locations to be occupied by the aircraft under each plan. Converting plan commands into route maps is costly; however, this cost is offset by caching the results in the ROUTE slot of the plan schema. Hence, each plan undergoes expansion only once. The route map is then compared to similar maps of the projected or known plans of other aircraft in the airspace. Maps are compared using an intersection search that requires maintenance of a 36-square-mile window around each aircraft. Detected conflicts trigger annotations of a plan's problems and utility that are stored in the CONFLICTS, CONFLICTSUM, and VALUE slots of the plan schema.

PLAN REFINEMENT

The plan generator refines initial plans whenever the evaluator detects conflicts in an initial plan. The patches we have implemented to date fall into three classes:

- Timing patches. These patches alter the time at which a plan's commands are executed without changing the commands themselves. For example, conflicts are often avoided by deferring or promoting (i.e., moving up in time) an altitude change command.

- Delaying patches. Inserting new commands in a plan to delay an aircraft's arrival at a particular point often prevents conflicts. For example, the insertion of a single turn command in an aircraft's plan can result in an 8-mile loop that will delay the aircraft's progress by several minutes.

- Course alteration patches. Conflicts can be avoided by charting a course alteration to avoid the conflict location. This usually requires deleting several course correction commands from the flight plan and replacing them with new ones.

These patch types are representative of more general replanning capabilities. For example, interpolating a loop is a kind of prerequisite insertion (Sussman, 1975), and course corrections amount to substitution of subgoals (Fahlman, 1974).

Each patch is represented as a schema with slots encoding the computations required to evaluate patch effectiveness and to modify a plan. These computations include tests for satisfaction of patch prerequisites, determination of where to insert new commands and/or where to excise old commands, and computations that actually perform the alteration to the existing plan. An abstraction of a patch that inserts a left-loop into a plan is shown below:

```
(PLAN-PATCH LOOP-LEFT
  TYPE           delaying
  DIRECTION      left
  PREREQUISITE   <the conflict point must not be too close to
                  an airspace boundary else the loop will take
                  aircraft out of airspace>
  INITIATEPOINT  <use the earliest point on the route that
                  satisfies prerequisites and is prior to
                  given conflict>
  DELETECOMMAND  nil
  ADDCOMMAND     <turn left until you are back at your current
                  heading>
  COST           low
  EFFECTIVENESS  high)
```

To instantiate a patch schema for use with a particular imperfect plan, the plan generator first attempts to satisfy the PREREQUISITES of the patch in the context of the plan. If this process is successful, the generator applies the heuristics in the INITIATEPOINT slot of the patch to select a point at which to insert a remedial command The specific commands are then copied from the ADDCOMMAND slot into the plan itself. In some cases (e.g., when deferring an altitude change), commands initially in the route must also be excised. The specifications for such deletions reside in the DELETECOMMAND slot. Finally, the ROUTE slot is updated to reflect the new flight path determined by the patched plan.

FLOW OF CONTROL DURING PLANNING

The plan generator posts the initial set of plans (usually three) in the world model. This triggers the evaluator to test the plans and post the results in the world model by filling the appropriate slots in the plans' schemata. If one of the initial plans is conflict-free, planning terminates and the controller begins executing the plan. If conflicts remain in all initial plans, patching must be attempted for one or more of them.

The patching process is best viewed as a search involving the plan generator and evaluator as co-routines. The generator iteratively expands a plan tree for the aircraft in the world model, using possibly flawed initial plans as the parent nodes. Offspring are generated through the application of one or more patches to the initial plans. Patches are applied to copies of the parent plan rather than to the original plan itself. Hence, the modified offspring are distinct data structures. Whenever a new plan is posted in the world model, the evaluator criticizes the plan and posts the results of its critique.

The generator selects plans for expansion (i.e., patch application) according to which current plan has the highest value in its VALUE slot. This value reflects the number of conflicts remaining to be resolved and the length of the flight path. Once a plan has been selected for patching, the plan generator applies only patches that generate better offspring (i.e., have higher VALUEs) than their parents. This heuristic results in depth-first/best-first searches, since offspring plans are always better than their parents. Planning terminates whenever one offspring plan has no conflicts or when the plan space is exhausted--that is, when no plan for the aircraft has a set of patches that remove all conflicts. In this case, the evaluator selects the plan with the highest VALUE for execution.

In the current implementation, searches typically converge quickly on a solution. Rate of convergence is governed by the density of solutions in the problem space (an inverse function of the number of active aircraft), the branching factor of the plan tree, and the depth of the plan tree. Both the breadth and the depth of the tree are limited. The branching factor is limited by the number of patch types known and by the fact that particular patches do not satisfy all prerequisites for application in a given situation. The depth of the tree is strictly limited by the number of conflicts found in initial routes.

VI. DISTRIBUTED PLANNING ARCHITECTURES

We are currently exploring techniques for distributed planning in several versions of the object-centered architecture. Two goals motivate the consideration of architectural variants: First, we want to develop a kernel design for a distributed planner that is robust across variations in the architectures in which it is embedded. Second, we want to understand the value of cooperation, the types of cooperation possible in a distributed environment, and the difficulties of achieving them.

We present here the structure of four distinct object-centered variants, and in Section VII we discuss their relative performance. The first two variants exemplify cooperation without bargaining or negotiation. In the first, the use of common planning rules and automated inference obviates the need for communication of plan intentions. In the second, aircraft communicate their plans but not their replanning requests. The third and fourth variants involve cooperative planning using different control regimes.

OBJECT-CENTERED AUTONOMOUS--NO COMMUNICATION

In the most restricted form of cooperation, aircraft plan autonomously without communication. Cooperation here is "culturally regulated," rather than "interpersonally interactive." Thus, in this architecture we excise the communicator from the AUTOPILOT kernel. In lieu of obtaining flight plans from other aircraft, AUTOPILOT, via the sensor, infers their plans from altitudes, bearings, and nearest exit fixes or airports along their current flight paths.

Due to the uncertainty associated with such extrapolation, the sensor must continually monitor the radar returns and the world model to detect changes in aircraft locations and violated assumptions about their flight plans. Updating the hypothesized flight plans triggers new conflict-detection checks by the evaluator. If new conflicts are predicted, the planner attempts to patch the current plan to avoid them. If the attempt is unsuccessful, the planner dynamically replans a new route. Effective cooperation is achieved through the use of global "rules of the road" and precedence rules, like those currently used by operators of small, visually controlled aircraft and boats.

OBJECT-CENTERED AUTONOMOUS--LIMITED COMMUNICATION

This variant differs from the preceding one in that the aircraft can request plans from other aircraft. Their intentions can therefore be posted with certainty, and their route maps can be accurately modeled rather than merely estimated. This version of AUTOPILOT therefore requires the communicator, communications channels, and protocols. By proscribing negotiation among aircraft, we place the burden of maintaining aircraft separation solely with the aircraft attempting to formulate a plan. Thus, as each new plane enters the airspace, it must develop a conflict-free plan with respect to the fixed flight plans of other aircraft already in the airspace. (When two or more aircraft enter simultaneously, planning order is determined by the alphabetical order of the aircraft identifiers.) Such an architecture should support effective planning only when the problem space is dense in solutions--that is, when a conflict-free plan can be produced regardless of the number and routes of other aircraft in the airspace.

Figure 3 illustrates the control structure of this version. When AUTOPILOT is assigned to a new aircraft, the sensor posts other aircraft locations, and the communicator collects and posts the flight plans for these aircraft. Initial plan generation by the planner may be interleaved with the functions of the sensor and communicator. The evaluator simulates the outcome of plan execution with respect to other aircraft locations and plans. If necessary, the planner attempts to patch the plan to eliminate specific conflicts detected by the evaluator. If the plan cannot be patched, the planner will attempt to generate a new plan. When either a conflict-free plan or the best available plan is posted as final, the controller monitors execution of the plan. (For simplicity, we have omitted the control and replanning feedback loops in Fig. 3).

In general, the utility of these autonomous versions of the object-centered architecture depends on several attributes of the problem space and task domain. First, autonomous planning, with or without plan communication, should succeed only when the problem space is dense in solutions. To develop a conflict-free plan independently, an aircraft must have more freedom, in terms of available airspace, than constraints on the locations it can occupy without conflict. Second, autonomous planning is preferred over cooperative planning when the cost (in time or resources) of local inferencing and planning is less than the cost of communications, negotiation, and coordination.

Introducing negotiation into AUTOPILOT's planning behaviors entails both costs and benefits. Inter-aircraft cooperation is desirable because the conflicting aircraft may have different options for resolving the conflict. One aircraft may discover a simple patch for its

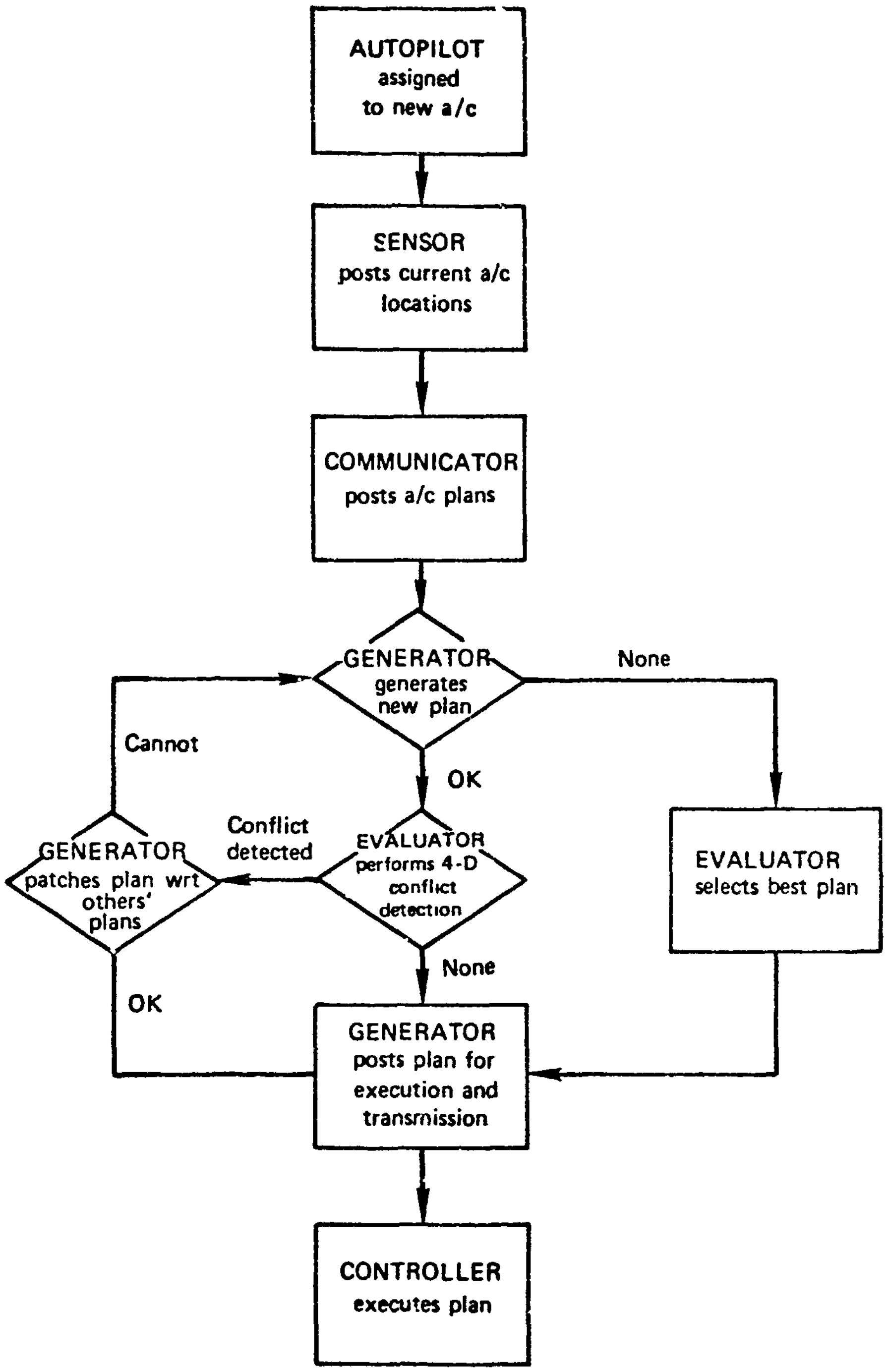
AUTOPILOT assigned to new a/c
SENSOR posts current a/c locations
COMMUNICATOR posts a/c plans
GENERATOR generates new plan
None
OK
Cannot
GENERATOR patches plan wrt others' plans
Conflict detected
EVALUATOR performs 4-D conflict detection
EVALUATOR selects best plan
None
OK
GENERATOR posts plan for execution and transmission
CONTROLLER executes plan

Fig. 3 — Control structure for the object-centered autonomous architecture

plan, while it may be impossible for another aircraft to remove the conflict in its plan.

However, complications arise from the need to synchronize local replanning activities. For example, assume that A has a route that conflicts at p1 with B and at p2 with C. Suppose that A can patch its plan to remove its conflict with C but must rely on B to replan to remove their mutual conflict. B cannct assume that A's plan will remain fixed, since A is patching its plan to accommodate C. In general, different conflicts (subproblems) may not be independent, and local planning cannot guarantee a globally satisfactory plan. Thus, cooperation through negotiation and communication requires effective _coordination regimes_. The following two architectural variants embody two very different techniques for such coordination. In each case, requests for cooperation are initiated by an aircraft that fails to find a conflict-free plan for itself.

OBJECT-CENTERED HIERARCHICALLY COOPERATIVE

In the hierarchically cooperative variant, the aircraft currently in planning mode (say A) becomes an explicit coordinator of the attempts to eliminate conflicts from its plan by local planning. Figure 4 illustrates the coordination regime. A's evaluator first selects its best plan. The communicator then passes a message to another aircraft (say, B) that conflicts with A's plan. The message contains A's plan and requests that B patch its plan under the assumption that A will execute its plan. If B's return message contains a successful patch, A makes the same request of the next aircraft (say, C) with a predicted conflict. A passes both its plan and B's tentative patch. If C responds

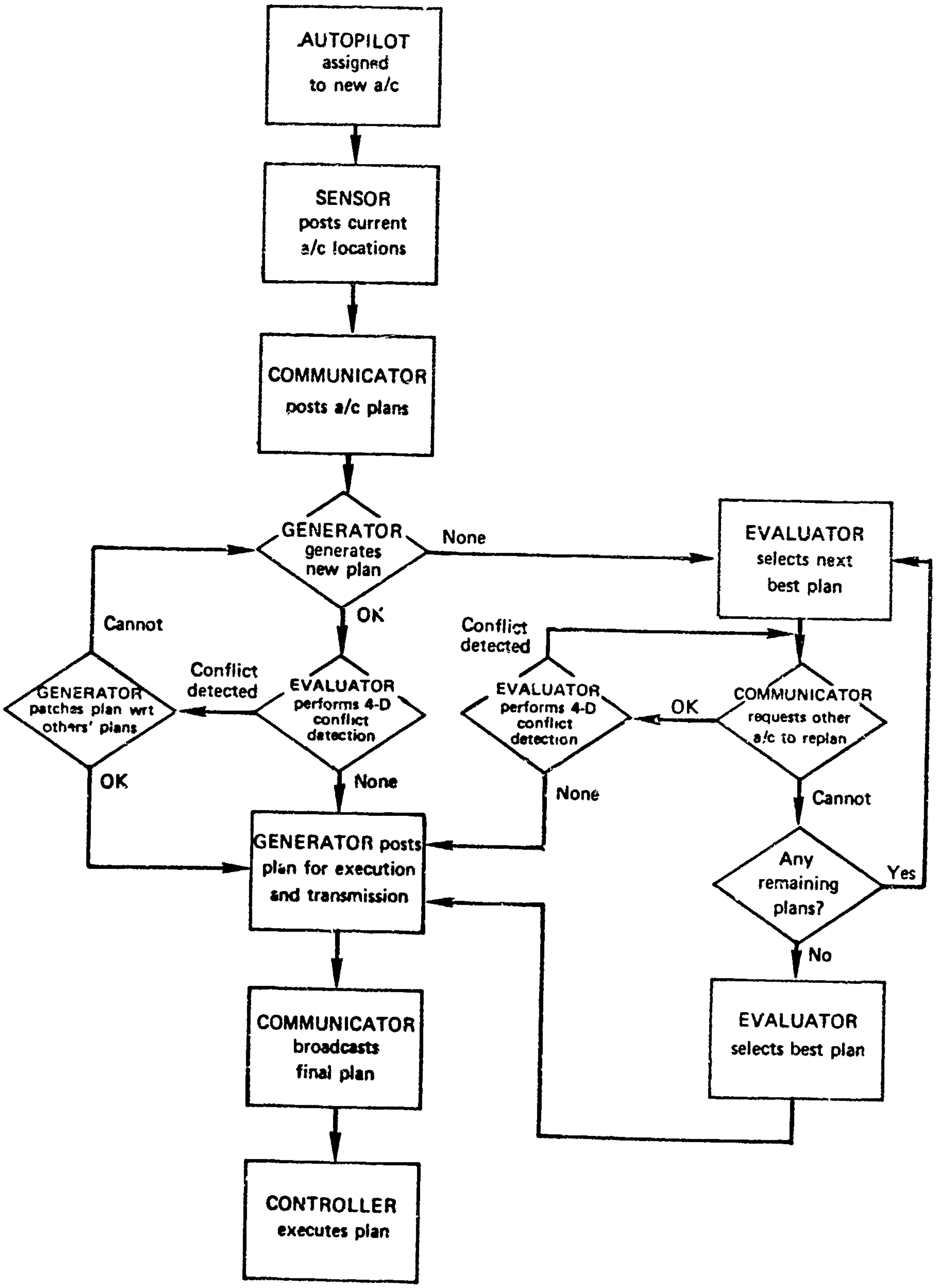
AUTOPILOT assigned to new a/c
SENSOR posts current a/c locations
COMMUNICATOR posts a/c plans
GENERATOR generates new plan
None
OK
EVALUATOR selects next best plan
Cannot
Conflict detected
EVALUATOR performs 4-D conflict detection
GENERATOR patches plan wrt others' plans
OK
None
Conflict detected
EVALUATOR performs 4-D conflict detection
OK
COMMUNICATOR requests other a/c to replan
None
Cannot
GENERATOR posts plan for execution and transmission
Any remaining plans?
Yes
No
COMMUNICATOR broadcasts final plan
EVALUATOR selects best plan
CONTROLLER executes plan

Fig. 4 — Control structure for the object-centered cooperative architecture

that it cannot patch under the given constraints, then A's evaluator will abandon this plan, select its next best plan, and the communicator will begin the negotiation process again.

OBJECT-CENTERED ASYNCHRONOUSLY COOPERATIVE

The same type of cooperation may be achieved through asynchronous, parallel replanning efforts. In this case, the planner requiring assistance does not dictate a particular, favored plan. Rather, the planning aircraft (A) broadcasts its set of potential plans to all aircraft in the conflict set (B, C, etc.) but sends no constraints to these aircraft concerning what assumptions they must adopt regarding A's or the others' patches. Each of these aircraft then attempts to patch its own plans to remove the conflicts predicted between it and A. Solutions are communicated to A as tentative plan revisions.

When B returns a plan to A that removes a common conflict, B also sends the assumptions under which it generated the solution--that is, the plan for A that B assumed in its revision. A must maintain a record of all proposed partial solutions and halt the asynchronous replanning process when (1) it has received a complete set of conflict elimination patches for one of its potential routes and (2) the proposed patched plans of the other aircraft do not conflict with each other.

Such cooperation accelerates the planning process by exploiting the parallel processing capabilities of multiple aircraft. When numerous pairwise conflicts must be resolved, the sequential solution method entailed by hierarchical control may require too much time to converge on a solution. However, in the asynchronous cooperation regime, speed is achieved at the cost of additional bookkeeping and evaluation by A.

VII. SYSTEM PERFORMANCE

We have implemented the limited-communication autonomous variant and the hierarchically cooperative variant of AUTOPILOT in INTERLISP on a DEC-2060 at Rand. They communicate over the ARPANET with the ATC simulation, a C program residing on a PDP-11/780. These variants differ only in the architecture in which AUTOPILOT is embedded, not in the planning or sensing capabilities of each aircraft. Table 1 presents performance data for these two architectures in simulation runs that varied airspace density. Each simulation run presented exactly 26 aircraft, distributed randomly in time, to be controlled. Airspace density was manipulated by varying the duraton of simulation runs--the shorter the duration, the greater the average density.

Both architectures perform with low error rates on simulation runs in low- to medium-density airspaces (i.e., 50- to 60-minute runs). In high-density airspaces (i.e., 30- to 40-minute runs), the hierarchically cooperative variant outperforms the autonomous system. This reflects the additional planning options that can be considered in cooperative architectures and that are required when air traffic is heavy.

Table 1

MEAN NUMBER OF UNRESOLVED CONFLICTS
PER SIMULATION RUN

	Simulation Duration (Min)			
Version	30	40	50	60
Autonomous	15.2	10.5	5.0	4.2
Cooperative	12.5	8.0	4.7	4.0

VIII. CONCLUDING REMARKS

We have illustrated several methods for distributing planning responsibility among multiple processors working toward a common set of objectives. Clearly, the object-centered architectures we have discussed are illustrative rather than exhaustive. In future work we will implement and evaluate the performance of other architectures and other variants on the object-centered architecture. In so doing, we will emphasize the development of more sophisticated bargaining methods and communications protocols. We also hope to determine how dense in solutions a problem space must be to utilize each of our developed architectures successfully.

In order to demonstrate and compare our candidate architectures, we have introduced several simplifications to our distributed planning environment. These include (1) simulation of multiple planners by a single planning program, (2) error-free communications, (3) limited route planning and revising heuristics, and (4) complete cooperation with no competition among different aircraft. Our future work will remove these simplifications from the task environment. In particular, we plan to achieve true distribution by demonstrating multi-processor control of real autonomous vehicles.

Our current work also addresses goals extending beyond a repertoire of domain-specific planning and patching techniques. The object-oriented programming techniques we have developed suggest a general framework for functionally distributed, communicating planners. At the same time, we currently know more about how to model cooperation than we do about what cooperation should be modeled. We still lack a theory of cooperation that would provide answers to questions such as, When should I request a plan from another? How much effort should I expend in planning before requesting another to replan? Under what conditions should objects plan for others in addition to themselves? Such questions are at the heart of effective cooperation in many distributed-planning domains.

REFERENCES

Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R. The Hearsay-II speech understanding system: Integrating knowledge to reduce uncertainty. Computing Surveys, June 1980.

Fahlman, S. E. A planning system for robot construction tasks. Artificial Intelligence, 5, 1974, 1-50.

Hayes-Roth, B., Hayes-Roth, F., Rosenschein, S., & Cammarata, S. Modeling planning as an incremental, opportunistic process. Proceedings of IJCAI-79. August 1979, pp. 375-383.

Hewitt, C. Viewing control structure as patterns of message passing. Artificial Intelligence, 8, 1977, 323-364.

Kahn, K. Director Guide, MIT AI Lab. Memo 482, June 1978.

Kay, A. A personal computer for children of all ages. Proceedings of the ACM National Conference, August 1972.

McArthur, D., and Sowizral, H. An object-oriented language for constructing simulations. Proceedings of IJCAI-81, August 1981.

Rucker, R. Automated enroute ATC (AERA): Operational concepts, package 1 description, and issues. MTR-79W00167, The Mitre Corporation, McLean, Virginia, 1979.

Sussman, G. A computational model of skill acquisition. New York: American Elsevier, 1975.