AUTOPILOT: A DISTRIBUTED PLANNER FOR AIR FLEET CONTROL*

Perry W. Thorndyke, Dave McArthur, and Stephanie Cammarata

The Rand Corporation
1700 Main Street
Santa Monica, CA 90406

ABSTRACT

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.

1. INTRODUCTION

Distributed 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 cooperating processors, and the discovery and implementation of planning techniques to be used by each processor. We have undertaken both activities in an effort to develop methods for distributed control of aircraft moving through an air traffic control sector. This task domain has permitted us to investigate three important questions concerning cooperation:

(1) What are the computational costs and benefits of different architectures for distributing planning functions?

(2) How should distributed planners cope with incomplete and errorful 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.

(3) What are the pragmatics of cooperation? Distributed planning should be superior to centralized planning only if methods can be devised for exploiting the resources of the multiple processors. These methods must confront the tradeoffs between local planning and requests for cooperation, and between inferring intentions and requesting information from others.

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 an aircraft by cooperating with virtual clones of itself, each of which is assigned to and controls a different aircraft. In this paper, 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.

2. THE ATC TASK DOMAIN

The task environment for AUTOPILOT is provided by a real-time air traffic control (ATC) simulation. Figure 1 illustrates a portion of 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 ("V and "#"), and a navigation aid ("*") through which aircraft can be vectored. During each run of the simulation, twenty-six 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 Figure 1) portrays the locations of all aircraft in the airspace, their identifiers, and their altitudes in thousands of feet (e.g., A5, X6). Every fifteen seconds the display is updated and the aircraft move one mile (to an adjacent "." or ",") in the direction of their current bearing.

* This research was sponsored by the Defense Advanced Research Projects Agency (DOD) under Contract No. MDA903-78-C-0029. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

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 current location (or origin, for pending aircraft), its destination, its altitude (in thousands of feet), and its bearing. For example, aircraft R, will enter the airspace in one tine-step of the simulation at infix location 2, heading northwest at an altitude of 6000 feet. Its destination is exit fix 0. A potential route for R would take R northwest to "*" and then north to 0.

Successful control of aircraft requires issuing commands to navigate planes to their desired destinations and maintaining at least three miles of horizontal separation or 1000 feet of vertical separation between any two aircraft. 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. . . .       Ap ,->2 5  SE
. A5. . , . . . , . . . . 1 Rj 2->0 6  NW
. . . , . . , . . . . . .
. . . , . , . . . . . . .
. . . . * . . . . . . . .
. . . , . , . . . . . . .
. . . , . . , . . . . . .
. . . , . . , . . . . . .
. . . , . . . . . . . . .
3 . . . , . . . , . . . 4
. . . . , . . . . , . , .
. . . . % . . . . # . . .
. . . . , . . . . , . , .
1 . . . 9 . . . 5 . . . 2
```

Figure 1. Output from the ATC simulation.

3. IBS DISTRIBUTION OF PUNNING EFFORT

In our simulation, planning responsibility is distributed among the aircraft. 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. Whenever a new aircraft appears in the airspace, a new AUTOPILOT clone is created and pertorms all planning and cooperation for that aircraft as it navigates from its origin to destination.

All AUTOPILOT clones are behaviorally identical and can be viewed as virtual copies of a generic ATC planner. The structure of AUTOPILOT resembles that of an independent actor (1), or object, as in SMALLTALK |2), DIRECTOR |3], or ROSS (4). 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.

4. THE AUTOPILOT DESIGN

AUTOPILOT contains a design kernel common to all 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 (5, 6), performance of these experts is triggered by particular conditions in the World

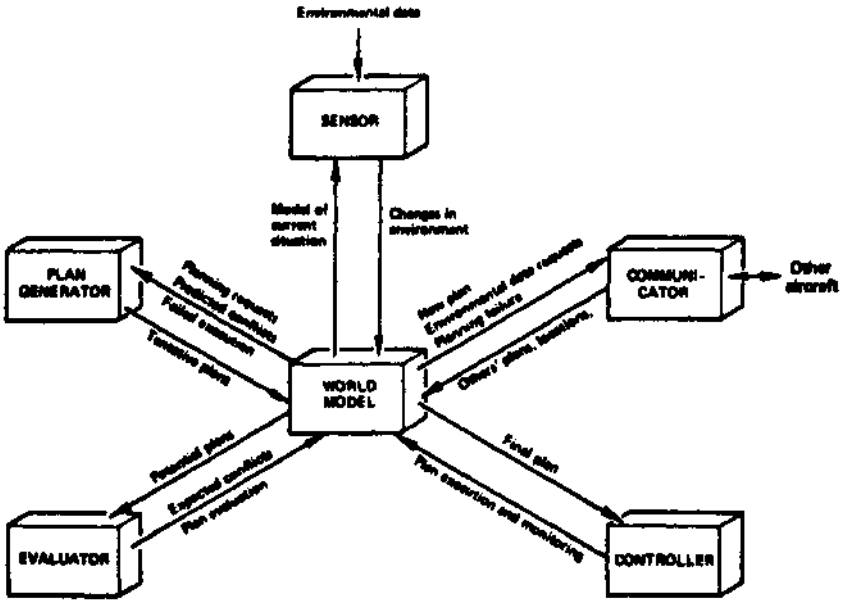

Fig. 2—Structure of the kernel planner

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 fifteen seconds. By comparing new displays to 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 origin to its destination. 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 to either 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 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.

5. INCREMENTAL PUNNING

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 Figure 1 is

```
(PLAN008
  AIRCRAFT     R
  COMMANDS     (a4 :0 a5)
  CONSTRAINTS  ((12 13) (12 13) (4 1))
  ROUTE        ((12 13 6 285) (11 12 6 300) ...)
  CONFLICTS    2
  CONFLICTSUM  ((X (6 7))) (X (4 16)))
  LENGTH       13
  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 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). Some slots in the plan schema are filled during plan generation, some during evaluation, and other during plan patching. The following paragraphs discuss these processes in more detail.

5.1. Plan Generation

AUTOPILOT produces route plans incrementally by planning approximately. The Plan Generator first produces with minimal effort a few standard routes from infix to the desired destination 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. 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 conflicts. This incremental approach to planning has four advantages. First, it emphasizes the 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.

5.2. 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 this 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.

5.3. Plan Refinement

The Plan Generator refines initial plans whenever the Evaluator detects conflicts in every initial plan. The patches fall into three classes. Timing patches alter the time at which a plan's commands are executed without changing the commands themselves (e.g., by deferring or promoting an altitude change command). Delaying patches insert new commands in a plan to delay an aircraft's arrival at a particular point (e.g., looping). Course alteration patches insert a detour to avoid the conflict location. This requires deleting several 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 [7], and course corrections amount to substitution of subgoals [8].

Each patch is represented as a schema with slots encoding the computations required to evaluate patch effectiveness and modify a plan. The following presents an abstraction of a patch that inserts a left-loop into a plan:

```
(PLAN-PATCH LOOP-LEFT
TYPE            delaying
DIRECTION       left
PREREQUISITE    <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 that
                 satisfies prerequisites and is
                 prior to conflict>
DELETECOMMAND   nil
ADDCOMMAND      <turn left 360 degrees>
COST            low
EFFECTIVENESS   high)
```

To instantiate a patch schema for use with a particular plan, the Plan Generator attempts to satisfy the PREREQUISITES of the patch in the context of the plan. If successful, the Generator applies the heuristics in the INITIATEPOINT slot 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 slot DELETECOMMAND. Finally, the ROUTE slot is updated to reflect the new flight path entailed by the patched plan.

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 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 implementations, 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 limited by the number of conflicts found in initial routes.

6. DISTRIBUTED PUNNING ARCHITECTURES

We are currently exploring techniques for distributed planning in four versions of the object-centered architecture. The first two variants exemplify cooperation without negotiation. In the first version, the use of common planning rules and local inference obviates the need for communication of plan intentions. In the second, aircraft communicate their plans but not replanning requests. The third and fourth variants employ cooperative planning using different control regimes. The following sections describe these four variants and the modifications in the kernel planner they entail.

6.1 Object-Centered Autonomous --No Communication

In this most restricted form of cooperation, aircraft plan autonomously without communication. Thus, in this version we excise the Communicator from the AUTOPILOT kernel. In lieu of obtaining flight plans from other aircraft, 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 hypothesised 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 the new conflict(s). If 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 used by operators of small visually-controlled aircraft, boats, and automobiles.

6.2. Object-Centered Autonomous-- Limited Communication

In this version an aircraft can request plans from other aircraft. Intentions can therefore be posted with certainty and route maps accurately

modeled rather than merely estimated. This version of AUTOPILOT therefore requires the Communicator, communications channels, and protocols. Proscribing negotiation among aircraft places the burden of maintaining aircraft separation on the aircraft attempting to formulate a plan. 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.

The Sensor first 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 conflicts detected by the Evaluator. When either a conflict-free plan or the best available plan is posted as final, the Controller monitors execution of the plan.

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--that is, so long as it is possible to produce a conflict-free plan regardless of the number and routes of other aircraft in the airspace. 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 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 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 cannot 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 different techniques for coordination. In each case, requests for cooperation are initiated by an aircraft that fails to find a conflict-free plan for itself.

6.3. Object-Centered Hierarchically Cooperative

In this version, the aircraft currently planning (A) orchestrates the attempts to eliminate conflicts from its plan. Figure 3 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 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.

6.4. 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 concerning what assumptions they must adopt regarding A's or the others' patches. Each aircraft simultaneously attempts to patch its own plan to remove 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 replanning process when (a) it has received a complete set of conflict elimination patches for one of its potential routes, and (b) 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 at A.

7. 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-AI. They communicate over the ARPANET with the ATC simulation, a C program residing on a PDP-11/70. These variants differ only in the architecture in which AUTOPILOT is embedded, but not in the planning or sensing

capabilities of each aircraft. Table 1 presents performance data for these two architectures. Both perform with low error rates on simulation runs in low* to medium-density airspaces (i.e., 50-60 minute runs). In high-density airspaces (i.e., 30-40 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

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

8. CONCLUSIONS

We have illustrated several methods for distributing planning responsibility among multiple processors working toward a common set of objectives. 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 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 our task environment. In

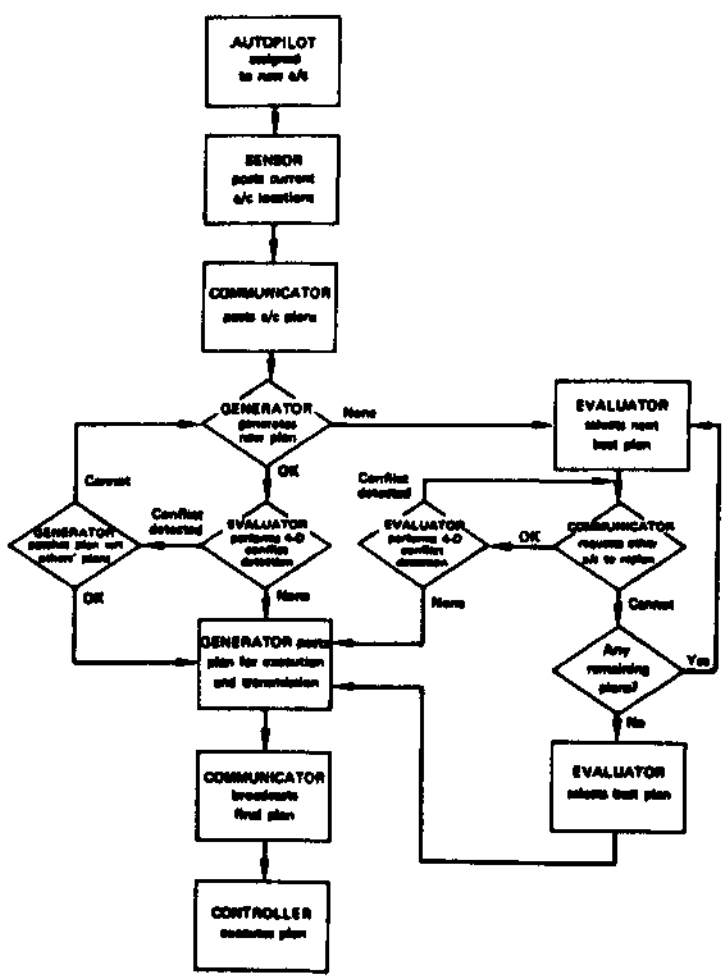

Fig. 2 — General structure for the object-centered cooperative architecture

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. In particular, 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 about what cooperation should be modeled. We still lack a theory of cooperation that would provide answers to questions such as: When should 1 request a plan from another? How much effort should 1 expend 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.

ACKNOWLEDGMENTS

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

REFERENCES

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

|2) Kay, A. A personal computer for children of all ages. Proceedings of the ACM National Conference, August, 1972.

|3] Kahn, K. Director Guide, MIT AI Lab. Memo 482, June, 1978.

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

[5] 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.

[6] 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.

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

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