Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

INTERACTIVE EXPERIMENT PLANNING TO CONTROL KNOWLEDGE-BASED SIMULATION

Robert E. Frederking

Center for Integrated Manufacturing Decision Systems (CIMDS)
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

ABSTRACT

Object-oriented simulation provides a powerful and conceptually clear methodology for discrete-event simulation. However, planning and executing simulation experiments using an object-oriented simulator is still a difficult, complex task. We are investigating the use of an interactive, opportunistic, hierarchical planner to control the design and execution of such simulation experiments. Our planner currently builds a hierarchical experiment plan interactively, setting parameters for the simulator. Examples are presented of the planner's graphical interface, which can also provide the simulator with a user interface.

1 INTRODUCTION: OBJECT-ORIENTED SIMULATION

The contributions of artificial intelligence (AI) to simulation to date include a renewed emphasis on object-oriented models (which AI inherited from the SIMULA simulation language), improved interface technology, and intelligent decision support tools. This paper describes an initial effort to use AI planning and interface technology to plan and control the execution of simulation experiments in an object-oriented discrete-event simulator.

Object-oriented simulation models a system as a set of interacting objects (Roberts and Heim, 1988). These objects interact by sending messages to each other when events occur. These messages invoke locally defined methods that implement the behaviors the object is capable of exhibiting. The resulting encapsulation of the data and procedures concerning a single object in a single place simplifies the construction of complex systems and permits a great deal of re-use of software. Previous object-oriented discrete-event simulators using AI technology (or ''Knowledge-Based Simulators'') include KBS (Fox and Reddy, 1982) (Reddy and Fox, 1982), the ROSS system at Rand (McArthur, Klahr, and Narain, 1986), and SimulationCraft^{TM} (Sathi et al., 1986).

Using an object-oriented simulator can reduce or eliminate the programming otherwise required to produce a discrete-event simulation, if suitable libraries of customizable objects are available. However, it is still a challenging task to produce a complete and consistent model; set all necessary simulation parameters to correct, consistent, and statistically meaningful values; monitor the execution of the simulation; and analyze its results. It is also difficult for the user of such a system to keep track of what has been done so far and what still needs to be done while interactively setting up a simulation run.

We are attempting to remove these remaining hurdles to the facile use of object-oriented simulation by using the CORTES (Fox and Sycara, 1990), (Fox and Sycara, 1991) opportunistic planner to control the planning and execution of simulation experiments in the CARMEMCO (Frederking and Chase, 1990) enterprise model.

CARMEMCO is a model enterprise, developed at CIMDS to support research in Computer Integrated Manufacturing (CIM). It is an object-oriented model, built within the frame-based KnowledgeCraft^{TM} AI representation language. The specific model implemented currently is a lamp manufacturing enterprise. The CARMEMCO simulator is an object-oriented discrete-event simulator, implemented by methods attached to the objects in the CARMEMCO knowledgebase. This simulator is a spiritual descendant of SimulationCraft^{TM} and KBS, which were also built within KnowledgeCraft^{TM} and its ancestor, SRL.

Within SimulationCraft^{TM}, an attempt was made to achieve the same goals as our project using expert system technology. The rule-based planning approach implemented lacked good global control, however, causing the system to confuse the user, presenting many unrelated suggestions at the same time. Our hierarchical, opportunistic planning approach, described below, should retain flexibility while keeping the interaction with the user focused.

Using the CORTES planner, the user interactively instantiates the activities and parameters of special interest, at whatever abstraction level and in any order desired. The user then activates automatic planning, causing the planner to produce a complete and consistent simulation experiment plan around the user's selections (if possible). Eventually, the planner could serve as the simulator's entire user interface, setting up the model and running, monitoring, and analyzing the simulation, interacting with the user and continually presenting the user with the experiment's current status via the planner's graphical interface.

2 AI PLANNING

In order to understand the detailed description of our system, some background in AI planning is necessary. There is a long history of planning research in AI, going back at least as far as the STRIPS (Fikes and Nilsson, 1971) system at SRI. Planners, such as ours, which follow this traditional approach are referred to as "classical", or "generative", planners, to distinguish them from several more recent planning paradigms, which we will not describe here. In a classical planner, a *plan* consists of a graph of *steps*, representing actions that will transform a given initial state into a goal state. A *state* is simply a description of the world at a point in time, usually as a conjunction of logical predicates or an equivalent representation. Plans are produced by instantiating *operators*. An operator is simply a parameterized abstract action, with *preconditions* describing the prerequisites for the operator's action and *postconditions* describing the effects of the action. For even relatively simple problems, searching for the optimal sequence of operators to achieve a non-trivial goal is difficult, due to interactions between the preconditions and effects of different operators.

In the CARMEMCO model, the mechanism for describing processes (actions) is the *activity* object. *State* objects provide a mechanism for describing and tracking the changes that activities make. Each activity has a *pre-state* that describes what must be true in order for the activity to occur and a *post-state* that describes what will be true after the activity occurs. The activity-state network representation for processes and system states was originally developed for use in the Callisto project management project (Sathi, Fox, and Greenberg, 1985). The activity-state representation is currently used in several projects at CIMDS, including SAGE (Roth and Mattis, 1988), an intelligent human interface manager. Since this activity-state representation was already in use in the CARMEMCO system, it was the obvious choice for representing our simulation experiment plans.

In the development of planning, a number of properties have become widely recognized as important, such as domain independence, hierarchical planning, and non-linear planning. Another property that we believe is important, opportunism, is still controversial.

2.1 Domain Independence

For a planner to be interesting from an AI point of view, it must be domain independent. If a planner is not domain independent, it can be very difficult to tell what is happening, in particular, whether it succeeds in planning because of its planning capabilities or because of clever, domain-specific representational and program design decisions by its human creators. Domain independence helps guarantee that it is the planning representations and operations that are doing the work.

2.2 Hierarchical Planning

Hierarchical planning is necessary if a planner is to take on real-world scale problems. Solving a large problem from beginning to end at the finest level of detail is completely intractable, even with heuristic AI methods. This can be remedied by first solving the problem at a very abstract level, and then refining each of the abstract activities into a set of activities at a finer level of detail, repeating until a ground level is reached in which the activities are primitive domain actions. Hierarchical planning was introduced in the ABSTRIPS system (Sacerdoti, 1973), and is widely used in current planners, such as SIPE (Wilkins, 1988).

An important distinction must be made between the hierarchical planning in ABSTRIPS and in planners such as ours and SIPE. In ABSTRIPS, the so-called "abstraction levels" in the hierarchy are actually criticality levels for domain predicates. A plan is found involving only the most critical preconditions first. Then preconditions at the next level of criticality are considered, and so on, until all criticality levels have been considered. These "abstract" plans, however, are composed of single ground-level steps, albeit the most critical ones. In truly hierarchical planning, higher-level activities are abstractions of groups of lower-level activities and do not necessarily appear at all in the final, ground-level plan.

2.3 Non-linear Planning

When a planner is asked to achieve a goal state containing a conjunction of goals, the problem of non-linear planning arises. Early planners simply planned each conjunct in turn, and strung the resulting plans together, "linearly". This produces badly non-optimal solutions, and can fail on problems that are actually soluble. The traditional solution to this is to produce conceptually parallel solutions for the conjuncts, represented as a partially ordered plan (Sacerdoti, 1975). Since in reality there usually are interactions between some of the goals, "critics" are used to add ordering constraints to these plans where necessary to make them correct. This approach to non-linearity is cumbersome and very computationally expensive.

The CORTES planner incorporates a new approach to non-linearity, which is inspired by the work of Veloso (Veloso, 1989). She points out that the deep problem with "linearity" is not the linearity of the plan representation used, but the "linearity" of completely solving one goal before working on the next. This linearity was a deliberate heuristic used in the early planners, to reduce complexity. When "non-linearity" was introduced, it was in the guise of partially ordered plans. Unfortunately, the partially ordered representation makes the computation of the truth of even one predicate very expensive.

Veloso's approach is simply to use a linear representation for plans, but allow the goals and subgoals to be worked on in any order. Retaining the much simpler linear plan representation but allowing free selection of the next goal to consider does increase the size of the search space. However, the linear representation allows conceptually simpler and computationally cheaper basic operations, and consolidates all the planner's heuristics in a single place: the search space. It is a "vivid" representation for plans (Etherington et al., 1989), (Levesque, 1986).

2.4 Opportunistic Planning

Opportunistic planning is the ability to select an operator for addition to a plan, and instantiate some of its parameters, without specifying where it should go in the plan until later. This allows the planner to "jump around" from one subproblem to another, as human beings tend to do.

In contrast, STRIPS had a fixed search strategy based on subgoaling. When an operator was instantiated, STRIPS would next try to achieve any unachieved preconditions of that operator. If successful, the operator would be "applied", changing the state. If the new state did not match the goal state, a new means-ends analysis would be done.

Although it is not universally recognized as important, opportunism is essential for our experiment planning application, since we allow the user to interactively develop whatever part of the plan is of special interest, and then have the planner fill in the rest consistently. Note that opportunism implies the free selection of subgoals. It thus necessarily results in a "non-linear" system, in Veloso's sense.

The concept of opportunism was first described by Hayes-Roth and Hayes-Roth (Hayes-Roth et al., 1979), who produced a blackboard based planning system with a notion of opportunism. If the system heuristically realized that some operation would be necessary, it would be added to the plan, with ordering determined later.

3 SIMULATION EXPERIMENT PLANNING

As we have said, the CORTES planner's plans are represented as activity-state networks. Operators are abstract activities with additional slots indicating their planning variables, preconditions, and postconditions. A planning problem is represented as an initial state and a goal state connected by an empty high-level plan, representing that an as-yet-undetermined plan will produce the not-fully-specified goal state from the initial state. When the first operator is instantiated into a plan activity, its pre- and post-states are created from the initial and goal states, modified by the action the activity represents. The initial, highly oversimplified operator definitions for the simulation planning domain are shown in figure 1.

As steps are added to the plan, intermediate states are added between the steps as well, explicitly representing both the predicates known to be true at each point in the plan, and unsatisfied goals at that point. In non-opportunistic, non-hierarchical applications of the planner, each search step either adds a plan step to the plan or instantiates an uninstantiated variable. In hierarchical applications, possible search steps include "expand step" and "pop hierarchy". These expand a step in the current plan (thereby moving to a lower level) and return to the higher-level step containing the current step, respectively. In opportunistic applications, the action of adding a step to a plan is separated into "create step" and "insert step into plan here" steps, thus allowing a step and some of its parameters to be specified without specifying its location. Mistaken choices lead to backtracking and the use of another alternative.

The possible steps to add to a plan are determined by means-ends analysis (MEA) of the states in the plan. This produces a set of partially instantiated operators, each of which will achieve some goal that exists in a state with no matching predicate. In later steps the other variables in the operator are instantiated. This variable instantiation is also driven by an appropriate version of MEA. Taking either of these steps can add new predicates to a state.

```
;;; Define top-level simplan operators

(defop acquire-goal
    :vars ((x is-a simulation-goal))
    :posta ((simulation-goal x)))

(defop acquire-model
    :expansion (:sub-operators (add-order))
    :posta ((model-acquired)))

(defop model-validation
    :prec ((model-acquired))
    :posta ((model-validated)))

(defop experiment-planning
    :vars ((x is-a simulation-goal))
    :prec ((simulation-goal x))
    :posta ((experiment-planned x)))

(defop experiment-execution
    :vars ((x is-a simulation-goal))
    :prec ((experiment-planned x)
           (model-validated))
    :posta ((experiment-executed x)))

(defop experiment-analysis
    :vars ((x is-a simulation-goal))
    :prec ((experiment-executed x))
    :posta ((experiment-analyzed x)(experiment-done)))


;;; Define lower-level simplan operators

(defop add-order
    :super-operator acquire-model
    :vars ((part is-a manufactured-object)
           (quantity integer)
           (date integer)))
```

Figure 1: Definition of Simulation Planning Operators

Predicates added to a state are propagated forwards through any operators that do not affect them to the states on the other side, while goals are propagated backwards. Thus, a predicate exists from the point where it is created (by an operator or the initial state) up to the point where it is destroyed by another operator. Similarly, goals exist from the point at which they are created (by a precondition of an operator or the goal state) backwards to the point where they are fulfilled (by the creation of a matching predicate). A *difference* exists in a state when there is a goal that has no corresponding predicate. This representation creates a complete picture of why things are happening at any point in the plan.

This process is how we achieve "non-linearity", since the planner can select any open goal to work on next; they do not need to be subgoals of the same goal. There is no built-in enforcement of any search strategy through the currently unsatisfied goals.

There are currently four selectable search strategies: bounded depth-first search, breadth-first search, user-selection of the next step, and user-generation of the next step. User-selection of the next step is selection from steps suggested by MEA, while user-generation allows the user to select the next activity to add to the plan and its location without regard to MEA. Thus the full range of possible actions is made available. At any point in the search the planner can be switched from one strategy to another. User-selection and user-generation are useful in developing search heuristics and analyzing domain constraints in other applications, in addition to their use in man/machine planning systems such as this one.

We have implemented truly hierarchical planning. High-level operators indicate a list of suboperators that should be used to expand them. When hierarchical expansion of a node is called for, the planner is recursively invoked on the selected node, using its preceding and following states as the initial state and goal, and its suboperators as the available operators.

(Eventually there will also have to be a mechanism for mapping between high-level and low-level states.) Success at a lower level results in marking the parent activity as successfully expanded, and, if automatic planning is in use, returning to the next higher level. Under manual (user-generated) planning, the lower levels can be re-entered, and the user can return to the higher level at any time. If lower-level activities are only partially instantiated, and the user returns to a higher level and then invokes automatic planning, it will return to the lower level and finish instantiating the activities there in the course of its planning.

3.1 Interactive Planning as a User Interface Methodology

As mentioned above, the methodology of guiding simulation design, setup, and execution with an interactive, hierarchical, opportunistic planner can also provide the system with a user interface. The conceptual motion of the user through steps in the plan, and levels in the hierarchy, provides a natural framework for interaction between the user and the simulation.

Hierarchical planning serves as an effective means of varying the user interface's focus of attention and level of abstraction. By changing levels in the planning hierarchy, the planner zooms in on the details of the subtask the user is currently interested in, or zooms out to give the user a view of the whole problem at a less detailed level.

Although we intend to use the planner in this fashion, and it in fact behaves as if it were currently controlling the whole simulation, at the moment it only has control of the system during model acquisition, because it was inserted into a pre-existing simulation system. Although not conceptually difficult, significant revisions to the simulator interface will be required.

The initial version of our graphical interface is illustrated below. As has been said, plans can be built interactively (using user-generation), automatically (using bounded depth-first search), or a mixture of the two. The user interactively plans by clicking on graphical buttons representing each of the available operators. The instantiated operators in the current partial plan are shown in another window. The state of the interface after three such steps are inserted is shown in figure 2. Open variables in the instantiated operators can be bound by clicking on the instantiated operator in the plan: a menu then pops up, presenting the available choices.

When finished specifying a partial plan, the user clicks on ''Automatic Planning Search'', which fills in the rest of the plan using MEA. As the planner alters the plan, the changes appear in the plan window. The set of available operators displayed changes to show those currently under consideration due to MEA.

After several more steps, the ''Acquire Model'' operator is added to the plan. Since this operator has a hierarchical expansion, the planner expands it, by instantiating an ''Add Order'' operator. Currently, unbound numerical parameters can only be bound by querying the user. The query for the ''Quantity'' of parts in the order to be added to the factory model is shown in figure 3. The final state of the planner, after ''popping'' to the top level of the final plan, is shown in figure 4.

3.2 Details of the Planner's Operation

The MEA for variable instantiation mentioned above solves a common problem in planners. It looks for a variable instantiation that does *not* create a precondition that does *not* exist in the preceding state. For example, in the ''blocks-world'' domain (a commonly-used simple planning domain of stacking blocks), if the planner has the goal of clearing a block off of block C, it generates a partially-instantiated move starting on the top of block C, without specifying what to move or where to move it to (due to the way the operators are encoded, it doesn't know that it must move the block already on block C). Our MEA for variables will produce the block on top of C as the only choice for the block to be moved, since any other choice would create a precondition that does not already exist. Many planners (including earlier versions of this one) would stupidly try to move blocks onto C to have them available for moving off of C! If no such ''good'' instantiation exists, the planner reverts to the usual blind instantiation of all possibilities.

In order to implement the predicate propagation and difference anaylsis described above, it is necessary to do a simple form of dependency maintenance for individual predicates as they propagate through states. This allows the system to tell where a predicate came from, and thus discern the subgoaling structure of the plan. To implement this predicate source tracking, each predicate in each state is marked to indicate how it was introduced into the plan. When predicates are propagated to new states, they carry this information with them. When MEA is carried out, subgoals are created by comparing predicates without regard to their sources, but the subgoals thus created still indicate their sources. In other words, a subgoal indicates the need for a predicate without specifying where it should come from, but does indicate where the need for the predicate arose.

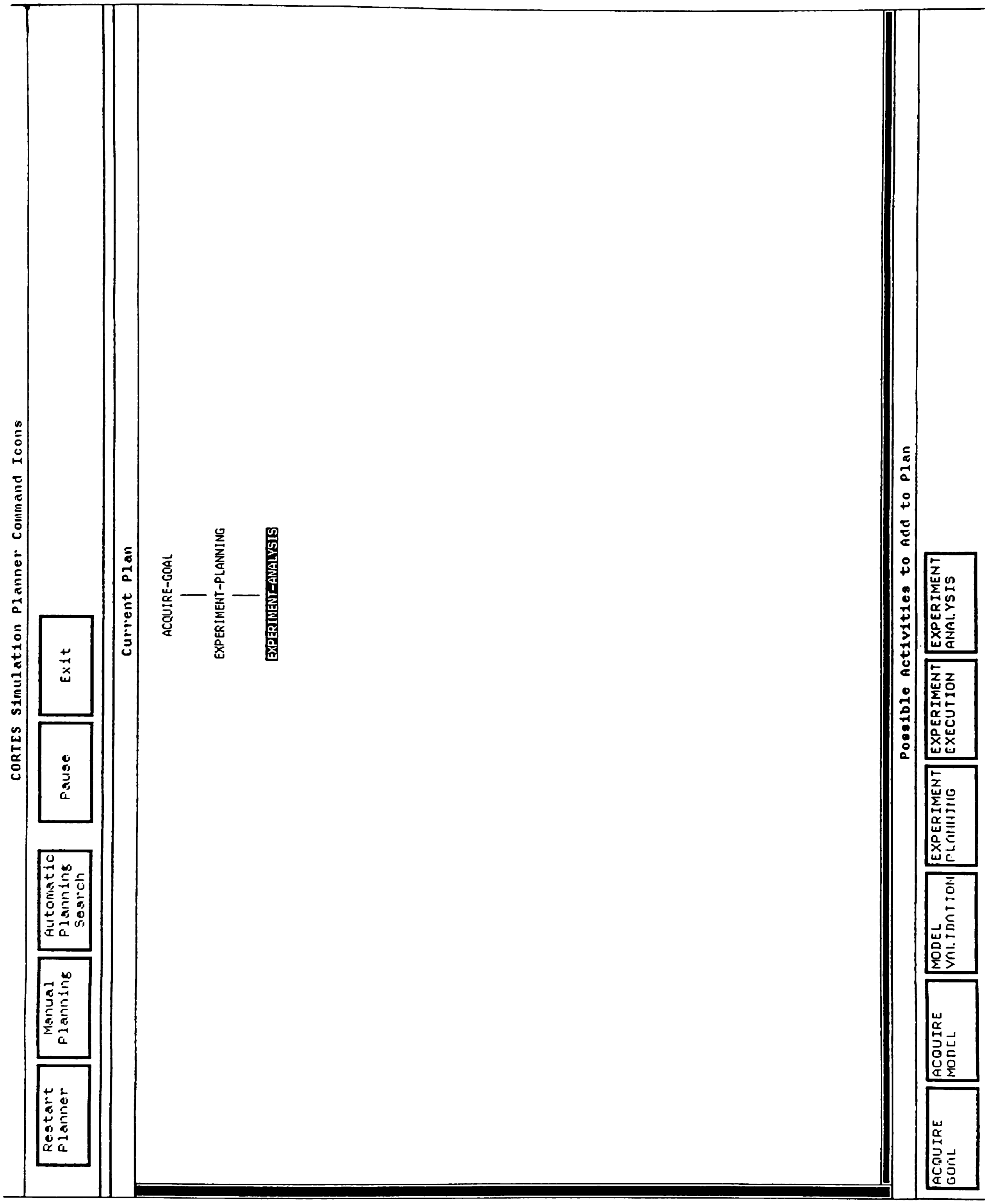
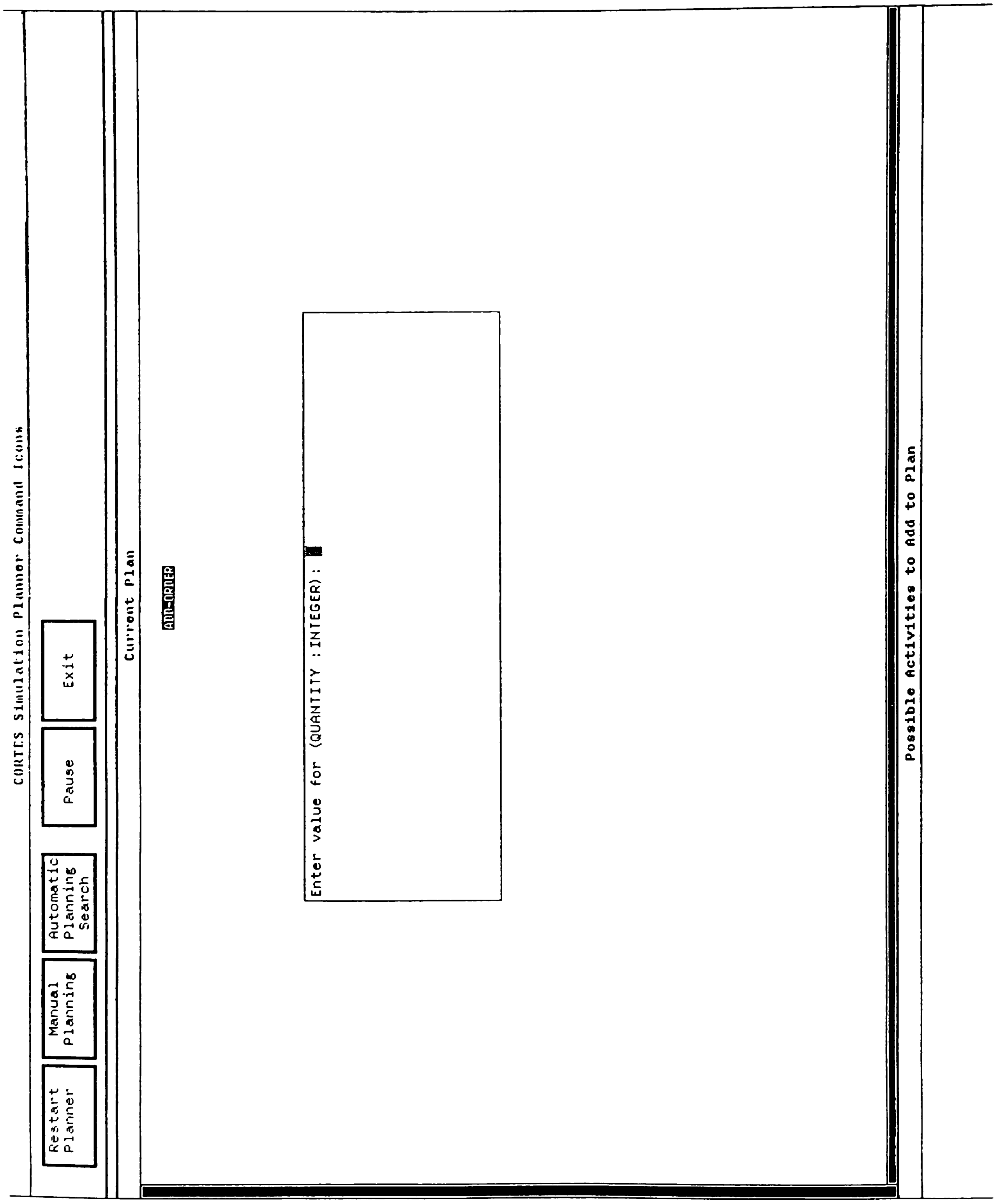
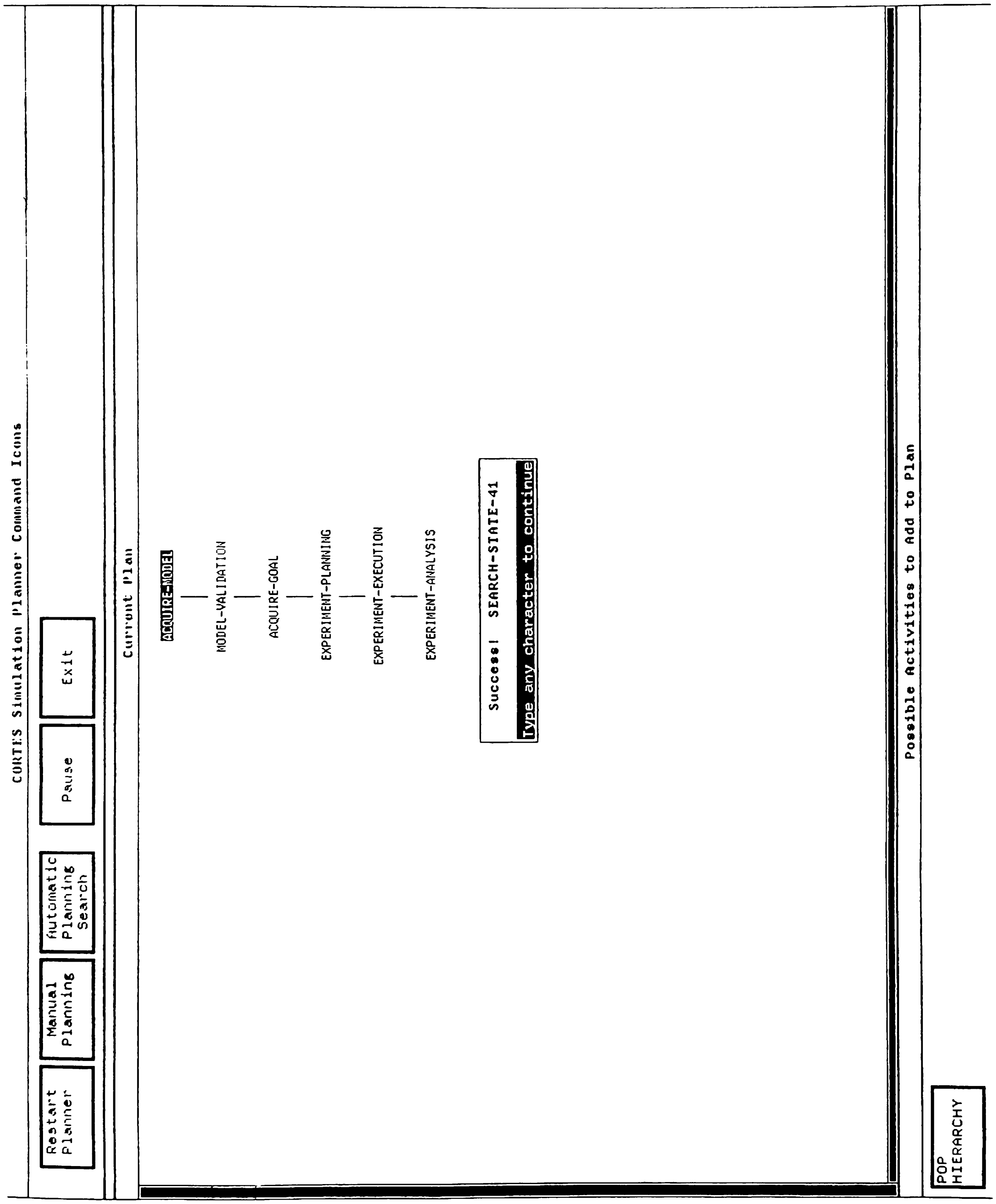


Figure 2: User Interface in Manual Planning Mode


CORTES Simulation Planner Command Icons
Restart Planner
Manual Planning
Automatic Planning Search
Pause
Exit
Current Plan
ADD-ORDER
Possible Activities to Add to Plan

Enter value for (QUANTITY :INTEGER):

Figure 3: Prompting for Input During Hierarchical Planning



Success! SEARCH-STATE-41

Type any character to continue

Figure 4: Final Plan Produced

The subgoal structure is also necessary if one is to detect subgoal loops or multiply achieved goals, or imitate the STRIPS search strategy. The first two are currently done. When the goal of a step is a subgoal (to any depth) of a step already taken to achieve the same goal, the new step's line of search is terminated. (Otherwise it could loop indefinitely, without any actual progress.) In multiply-achieved-goal elimination, if the same goal from the same source that the current step would achieve has already been achieved earlier in this search path, this path is terminated. The motivation for this is that there must be a shorter plan that only achieves the goal once, and this path could lead to an infinite loop similar to those in subgoal looping. We believe this is related to the well-known technique of "goal protection", where a planner will refuse to introduce an operator that would destroy a predicate between where it is created and where it is needed. This technique achieves the same results by different means.

Because operators can be introduced in any order, it is possible for the same plan to be arrived at via different search paths. The inefficiency this could cause has been remedied by maintaining a plan hash table. Every partial plan produced is hashed into this table. When a new search step is taken, the resulting plan is checked against the hash table. If it is found, the step is abandoned, since this partial plan is already being investigated somewhere else in the search tree. This has saved over 60% of the search space in deep searches.

Static preconditions can be included in our operators. This allows an operator to require that any of its instantiations satisfy certain predicates, without allowing the predicates to become new goals. Our encoding of the Tower of Hanoi makes use of this feature, since whether a disc can be set on another disc depends on its size, but no domain operator can alter a disc's size. Without static preconditions, useless subgoals would be produced to make one disc bigger than another. This feature works by causing the failure of a variable instantiation if any of the static preconditions are made false. Instantiation failures can also be caused by predicate propagation, if a predicate propagated to a state clashes with a postcondition of the step before that state.

Further details of the planner's operation, implementation, and other applications can be found elsewhere (Frederking and Chase, 1990), (Fox and Sycara, 1991).

4 FUTURE DEVELOPMENTS

Much further work remains to be done, in the simulator, the planner, and the graphical interface.

The simulator is currently at a very early stage of development. All important capabilities have been demonstrated, but only in a very narrow "vertical slice" through the system. Much development will be needed before interesting factory simulations can be carried out.

In the current version of the planner, the user must eventually specify a total ordering before starting depth-first search, since the depth-first search strategy used in automatic planning assumes totally ordered plans. Eventually it should be possible to have the planner create all total ordering possibilities that satisfy the plan's goals as search alternatives. The initial version also allows only one ordered set of activities; that is, there is exactly one totally ordered partial plan and a set of unordered activities. Later versions should allow more than one totally ordered partial plan, and should eventually allow partially ordered partial plans as well. The planner should also be extended to be able to reason about numerical parameters, rather than always prompting the user for them.

Although the graphical interface is currently quite usable, much remains to be done before it has all the desirable functionality. It would be very useful if the interface graphically indicated which high-level steps have previously been hierarchically expanded, and which have not, for example by coloring them green and red. Also, it would be helpful if the values of parameters that have already been bound were visible, at least optionally. As was said, the control of the overall simulation needs to be implemented. Finally, the interface currently does not handle full opportunism. When the graphical interface is in operation, the user must indicate the relative order of the plan steps as they are instantiated. This limitation is only temporary, resulting solely from the need for a new interface module to manage the screen locations of unrelated sets of plan steps, and to move them when their ordering is indicated later.

5 CONCLUSION

We have presented here an initial attempt to facilitate the design and execution of simulation experiments through the use of an interactive, opportunistic, hierarchical planner. The planner provides an orderly way to control the planning and execution of simulation experiments without overly restricting the user to a pre-established routine. In addition to planning the simulation and keeping track of its current state, the planner, through its graphical interface, can provide the simulator with a natural, coherent framework for its user interface.

ACKNOWLEDGMENTS

This research has been supported, in part, by the Defense Advance Research Projects Agency under contract #F30602-88-C-0001.

The CARMEMCO knowledgebase and simulator were designed and implemented by Lin Lawrance Chase. The graphical object-oriented user interface was based on a design and implementation by Joe Mattis, modified by the author. This project is based on a concept by Mark Fox.

KnowledgeCraft^{TM} and SimulationCraft^{TM} are products of Carnegie Group Inc., Pittsburgh PA 15222.

REFERENCES

Etherington, D.W., A. Borgida, R.J. Brachman, and H. Kautz. (1989). Vivid Knowledge and Tractable Reasoning: Preliminary Report. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* IJCAI-89.

Fikes, R.E. and N.J. Nilsson. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Proceedings of the Second International Joint Conference on Artificial Intelligence.* IJCAI-71.

Fox, M.S. and Y.V. Reddy. (1982). Knowledge Representation in Organization Modeling and Simulation: Definition and Interpretation. *Proceedings of the Thirteenth Annual Pittsburgh Conference on Modeling and Simulation.* University of Pittsburgh, School of Engineering, Volume 13.

Fox, M.S. and K. Sycara. (1990). Overview of CORTES: A Constraint Based Approach to Production Planning, Scheduling and Control. *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management.* ESPOM-90.

Fox, M.S. and K. Sycara. (1991). CORTES Project 1991 Annual Report. In preparation.

Frederking, R.E. and L.L. Chase. (1990). Planning in a CIM Environment: Research Towards a Constraint-Directed Planner. *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management.* ESPOM-90.

Hayes-Roth, B., F. Hayes-Roth, S. Rosenchein, and S. Cammarata. (1979). Modeling Planning as an Incremental, Opportunistic Process. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence.* IJCAI-79.

Levesque, H. J. (1986). Making Believers Out of Computers. *Artificial Intelligence, 30,* 81-108. Originally given as the "Computers and Thought" lecture at IJCAI-85.

McArthur, D.J., P. Klahr, and S. Narain. (1986). ROSS: An Object-Oriented Language for Constructing Simulations. In P. Klahr and D. Waterman (Eds.), *Expert Systems.* Addison-Wesley.

Reddy, Y.V. and M.S. Fox. (1982). Knowledge Representation in Organization Modeling and Simulation: A Detailed Example. *Proceedings of the Thirteenth Annual Pittsburgh Conference on Modeling and Simulation.* University of Pittsburgh, School of Engineering, Volume 13.

Roberts, S. and J. Heim. (1988). A perspective on object-oriented simulation. *Proceedings of the 1988 Winter Simulation Conference.* WSC '88.

Roth, S.F. and J. Mattis. (1988). Data Characterization for Intelligent Graphics Presentation. *Proceedings of the CHI '90 Conference.* Seattle, Washington: ACM.

Sacerdoti, E.D. (1973). Planning in a Hierarchy of Abstraction Spaces. *Advance Papers of IJCAI-73.* IJCAI-73.

Sacerdoti, E.D. (1975). The Nonlinear Nature of Plans. *Advance Papers of IJCAI-75.* IJCAI-75.

Sathi, N., M.S. Fox, V. Baskaran, J. Bouer. (1986). SimulationCraft^{TM}: An Artificial Intelligence Approach to the Simulation Life Cycle. *Proceedings of SCS Summer Simulation Conference.* Reno, Nevada.

Sathi, A., M.S. Fox, and M. Greenberg. (1985). Representation of Activity Knowledge for Project Management. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-7,5,* 531-551.

Veloso, M.M. (1989). *Nonlinear problem solving using intelligent casual-commitment* Technical Report. School of Computer Science, Carnegie Mellon University.

Wilkins, D.E. (1988). *Practical Planning, Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers.

AUTHOR BIOGRAPHY

ROBERT E. FREDERKING is a project scientist for the CORTES project in the Robotics Institute of Carnegie Mellon University. He holds a BS in Computer Engineering from Case Western Reserve University, and a PhD in Artificial Intelligence from Carnegie Mellon University. He has consulted for Carnegie Group Inc. and was a Research Engineer at Siemens Corporate Research Laboratories in Munich, West Germany. His research interests include knowledge representation and reasoning, natural language, and the foundations of computer science.