PANDORA – A Program for Doing Commonsense Planning in Complex Situations*

Joseph Faletti

Computer Science Division
Department of EECS
University of California, Berkeley
Berkeley, California 94720

Abstract

A planning program named PANDORA (Plan ANalyzer with Dynamic Organization, Revision, and Application) has been developed which creates plans in the commonsense domains of everyday situations and of a Unix** Consultant using hierarchical planning and meta-planning. PANDORA detects its own goals in an event-driven fashion, dynamically interleaving the creation, execution and revision of its plans.

1. Introduction.

Most early work in AI problem solving has used simple control structures to work in restricted or specialized domains (e.g., Fikes and Nilsson (1971), Newell and Simon (1972), Sussman (1975)), although some recent work has moved to common-sense domains (Rieger (1975), Hayes-Roth and Hayes-Roth (1979), Carbonell (1978), Cohen and Perrault (1979)) and more powerful control structures (Sacerdoti (1977), Genesereth (1978), Stefik (1980)).

We have developed a theory of planning described in Wilensky (1981) which suggests that the design of a planning program should include:

1. Shared knowledge with a planning story understander.
2. Use of common-sense domains.
3. Use of hierarchical planning and meta-planning.
4. Detection of its own goals in an event-driven fashion.
5. Dynamic interleaving of the creation, execution and revision of plans.

2. PANDORA – Commonsense Planning in Complex Situations.

PANDORA (Plan ANalyzer with Dynamic Organization, Revision, and Application) is a program incorporating the features described. It is implemented in PEARL, an AI programming language developed at Berkeley (Deering, Faletti, and Wilensky, (1981) and (1982)). PANDORA uses the same planning knowledge as the newest version of PAMELA, its story understanding counterpart, implemented by Peter Norvig. PANDORA and PAMELA also share an inference and frame-based memory package of knowledge and routines which perform all low-level processing of input and instantiating of frames.

* This research was sponsored in part by the Office of Naval Research under contract N00014-80-C-0732 and the National Science Foundation under grant MCS79-06543.

** Unix is a trademark of Bell Laboratories

3. Examples from Two Domains.

PANDORA has been applied to two commonsense domains. PANDORA's original domain was that of every day human situations. For example, one task PANDORA plans for involves the task of retrieving the morning newspaper when it is raining. Given the input:

; It is morning.
(TimeOfDay (Time Morning))

; It is raining.
(Weather (Object (Outside)) (Condition Raining))

PANDORA detects its normal morning goal of knowing what is going on in the world, modifies it to adjust for the rain, and produces the following plan:

; Put on a raincoat.
((PutOn (Actor (Ego)) (Object (Raincoat)))
; Go outside.
(PTrans (Actor (Ego)) (Object (Ego)) (To (Outside)))
; Pick up the newspaper.
(Grasp (Actor (Ego)) (Object (Newspaper)))
; Go inside.
(PTrans (Actor (Ego)) (Object (Ego)) (To (Inside)))
; Read the newspaper.
(Read (Actor Ego) (Object (Newspaper))))

That is, PANDORA figures out that she has to put on a raincoat before going outside, something that she does not normally do to retrieve the newspaper.

In order to generate this plan, PANDORA must

1. Notice that going outside this morning would get her wet and that this is a state she wants to prevent. This is accomplished by the Noticer and the Goal Detector.
2. Detect the resulting goal conflict between her plan of going outside to get the newspaper and the Stay Dry preservation goal. The Goal Detector does this.
3. Find a meta-plan which will find a way to alleviate this problem. This is done by the Plan Selector.
4. Execute this meta-plan which must find the Wear-Raincoat plan and modify the original plan. This is done by the Executor.

PANDORA is also being applied to the domain of using the Unix** operating system. The Unix domain was chosen so that PANDORA could be used as a problem-solver component of the natural language Unix Consultant UC (Wilensky (1982)) now being developed at Berkeley. UC will call on PANDORA whenever the question presented involves more complicated planning.

For example, if we tell PANDORA that PAMELA has a new electronic mail address by asserting

```
(MailAddress (Person PAMELA)
             (Where (Address (String ("kim:pamela")))))
```

into the data base, and later enter (Type (Actor Vi) (To Ego)

```
      (Message (String ("out of disk space"))))
```

when PANDORA is waiting for feedback from the write command in the editor, PANDORA generates and executes plans which result in the following steps being executed:

```
; Edit the Addresses file.
(StartUp (Program Editor))
(MakeChanges (File Addresses)
             (Changes (AddAddress ())))

; Attempt to write the file but fail.
(Type (Text (String (":w"
                    (FileName (File Addresses))
                    (Return))))))
(VerifySuccess))) ;— Reads error message from editor.

; Plan to get rid of the problem.
; Try to delete unneeded files but fail to find enough.
(DeleteUnneededFiles (Actor Ego))

; Temporarily save file in temporary space.
(Type (Text
      (String (":w"
              (FileName (File ("/usr/tmp/pandora")))
              (Return))))))
(VerifySuccess)))

; Save file more permanently.
(Mail (To Ego) (Content (File "/usr/tmp/pandora")))

; Ask system manager for more disk space.
(Mail (To System)
      (Content (Message
              (Request (MoreDiskSpace)))))
(Quit (Program Editor))
```

That is, PANDORA figures out first that she should edit her address book. When during the write command, she gets an "out of disk space" error message from the editor, she determines that she should try to delete some unneeded files and try the write command again. Failing this, PANDORA finds somewhere else to save the file, i.e., in temporary file space. However, since temporary files are not secure from removal by other users, PANDORA predicts eventual failure of this goal and finds a more permanent solution which is to mail the file to herself. This manages to save the file for the moment but does not get rid of the problem. PANDORA then generates a plan to acquire more disk space which is to ask the powers that be for more.

Note that for this example the above is the sequence of actions of PANDORA, rather than a plan generated in its entirety before execution. This is necessary because PANDORA must handle a goal conflict during execution rather than before, since the problem does not arise until then. This example also requires PANDORA to:

1. Resolve a goal conflict and then retry a plan. This is handled by the meta-plan doing the repair.
2. Handle a failure of the normal plan for getting more disk space by finding another plan. This is done by the Plan Selector.
3. Construct a temporary plan (that is, one likely to fail eventually) in order to carry out a more permanent one.

4. The Overall Structure.

There are three types of objects which PANDORA's control structure must deal with: external events, goals, and plans. The top level of control is simply a loop which deals with these in turn:

1. If there is an external event to process it is dealt with. This process might potentially involve the inference system, the Noticer and the Goal Detector.
2. If there is none, then plans are chosen for any unplanned goals. This is carried out by the Plan Selector and Projector.
3. If there are no goals to plan, then the plan at the top of the plan queue is executed. This is carried out by the Executor.

5. Inference, Noticing and the Goal Detector: Reacting to Input.

In the rain example, PANDORA reacts to external input representing the fact that it is morning and raining outside by detecting the frames and associated goals that apply to these situations. In this first stage, all of the significant work has been done by the low-level inference and frame invocation processes in conjunction with two important parts of PANDORA, the Noticer and the Goal Detector. The memory and frames package automatically watches for events that should invoke frames, choosing them based on the structure of each frame. (Exactly which pieces of a frame should be allowed to invoke it is still undecided but for PANDORA's situations the connections have been obvious so far.) Here the "morning" frame and the "rainy day" frame are invoked. Associated with each frame which describes a situation (as opposed to an action, event or state) is a list of goals that PANDORA normally has when that situation arises. The morning frame includes PANDORA's goal of knowing what is going on in the world.

Also associated with each frame is a set of inference rules which might apply when that frame is active. Hearing that it is raining outside invokes the "rainy day" frame which includes an inference rule to the effect that if someone goes outside they will get wet. In the case of remembering PAMELA's address, PANDORA has an inference rule which says that when it hears a friend's address, it should remember it.

6. The Plan Selector: Choosing and Installing Plans.

Whenever there is no more input, PANDORA's Plan Selector proceeds to plan for any goals it has. Considering that the main thrust of PANDORA is planning, the actual planning algorithm control structure is misleadingly simple for most goals. For each goal, this involves:

1. Choose the normal plan, if any.
2. Check it for conflicts with other plans by projecting its effects.
3. If it is all right, install it.

For example, PANDORA's normal plan for the "find out about the world" goal is to read the morning newspaper, which involves going outside, picking up the newspaper, and returning inside to read it. Her normal plan for remembering an address it to store it in its on-line

address book using the editor.

PANDORA always uses the normal plan for a goal unless it fails in some way or causes a goal conflict with some other goal. Since the emphasis of this work is on commonsense goals and plans, I consider mostly goals whose plans are well-known and require very little thought to select. Even for the more complicated situations of goal conflicts which are common, the above algorithm works.

One reason for this deceptive simplicity is the fact that most of the planning knowledge and therefore most of the planning algorithms are represented as meta-plans which accomplish meta-goals of the planning process. Thus the meat of the planning is buried in the knowledge base which must be relatively large for this type of planning. However, although it is large, it is quite broadly applicable to many classes of specific goals.

Meta-goals are treated just like any other goals. Thus, for example, PANDORA also looks for a normal plan for the Resolve Goal Conflict meta-goal detected during simulation of the Retrieve Newspaper plan (see the next section for more on this). In this case PANDORA finds and executes the meta-plan Replan which looks for a plan that avoids the conflict. Replan finds that wanting to go outside without getting wet can be accomplished by the plan of putting on a raincoat first. Replan is a meta-plan rather than just an ordinary plan because it involves using general planning knowledge to find a modification to the current plan.

7. The Projector and Noticer: Simulating the Chosen Plan.

PANDORA gives each plan a cursory check for conflicts by simulating the plan (currently only the top level of the plan is done), recording the effects of each step in a data base of events whose occurrence are projected into the future. These events are subject to the usual inference processes, but any results are recorded in this future data base.

For example, in the rain example, the planned act of going outside causes a problem. The effect of this act which is asserted into the future data base is that PANDORA expects to be outside in the near future. This causes the inference rule from the rainy day frame to infer that PANDORA will get wet.

Included in PANDORA's knowledge base are themes (collections of goal states organized under one property of an actor (Schank and Abelson (1977)) which organize sets of states which are to be maintained. One is the Preserve Health theme which includes the requirement that PANDORA remain dry and keep well fed. For each of these goals, the Noticer is informed that if any of these is violated, the Goal Detector should be informed. The Goal Detector will examine the state and generate a preservation goal.

Note that it is not good enough to simply notice that a state that was in the data base has changed. Instead, each state which is a goal state or a precondition of an intended plan is marked as being such. Then, each time a state change is inferred, such a mark is checked for and if it is so marked, the Goal Detector is informed.

If the culprit is an act the Goal Detector also knows that this is a goal conflict and generates the meta-goal of resolving this conflict. Meta-goals are treated like every other goal in PANDORA -- they are stacked and the normal planning process is performed on them.

Before handling a goal that is generated during simulation, PANDORA currently projects all of the steps of the currently proposed plan. However, note that after the simulation, the top level control structure implies that if there are still goals to be planned, the plan cannot be executed yet. In particular, any Resolve Goal Conflict meta-goals generated during the projection process must be planned for.

8. The Executor: Using the Plan.

Before PANDORA executes any regular plan, all currently active goals must have been planned for. If there are none, PANDORA chooses the next plan on the queue and executes it. The execution process involves two alternative steps. If the next plan to be executed has subplans, then these are installed in the queue, subject to the same projection process as plans in the original planning phase. If not, the plan is carried out by asserting its effects into data base.

One exception to this is made for meta-plans. Since they are normally part of the act of planning and not just acts to be planned, they must be executed immediately (after simulation to detect conflicts). For example, the meta-plan which installs the Put On Raincoat plan in front of the PTrans in the rain example is run immediately.

9. More on the Goal Detector.

Goals may be detected in PANDORA during most other processes. The ways that goals are detected may be summarized as follows:

1. Most situations have goals attached to them which PANDORA needs to plan for. In addition to our common morning goals, a good example of this is the set of goals which arise whenever friends come to visit.
2. Preservation goals may be detected whenever any statechange is asserted or projected into the data base.
3. Most goal interactions (both positive and negative) may be detected whenever any statechange is asserted or projected into the data base which contradicts a desired state from a goal or a precondition of a plan.

The current implementation of PANDORA concentrates on detecting this third kind of goal, arising from goal interactions.

10. References

Carbonell, J. 1978. *Computer Models of Social and Political Reasoning*. Ph.D. Thesis, Yale University, New Haven, Conn.

Cohen, P. and Perrault, R. 1979. Elements of a Plan-Based Theory of Speech Acts. *Cognitive Science*, Vol. 3, No. 3. 1979.

Deering, M., Faletti, J., and Wilensky, R. 1981. PEARL: An Efficient Language for Artificial Intelligence Programming. In the *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*. Vancouver, British Columbia. August, 1981.

Deering, M., Faletti, J., and Wilensky, R. 1982. The PEARL Users Manual. Berkeley Electronic Research Laboratory Memorandum No. UCB/ERL/M82/19. March, 1982.

Fikes, R. and Nilsson, N.J. 1971. STRIPS: A new approach to the application of theorem to proving problem solving. *Artificial Intelligence 2*, 189-208.

Genesereth, M.R. 1978. Automated Consultation for Complex Computer Systems. Ph.D. thesis, Harvard University.

Hayes-Roth, B. and Hayes-Roth, R. 1979. Cognitive Processes in Planning. RAND Report R-2366-ONR.

Newell, A. and Simon, H.A. 1972. *Human Problem Solving*. Englewood Cliffs, N.J.: Prentice Hall

Rieger, C. 1975. The Commonsense Algorithm as a Basis for Computer Models of Human Memory, Inference, Belief, and Contextual Language Comprehension. In *Theoretical Issues in Natural Language Processing*, R. Schank and B.L. Nash-Webber, (eds.), Cambridge, Mass.

Sacerdoti, E. 1977. *A Structure for Plans and Behavior*. Elsevier North-Holland, Amsterdam.

Schank, R.C. and Abelson, R.P 1977. *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Stefik, M.J. 1980. Planning and Meta-Planning -- MOLGEN: Part 2. Stanford Heuristic Programming Project HPP-80-13 (working paper), Computer Science Department, Stanford University.

Sussman, G.J. 1975. *A Computer Model of Skill Acquisition*. American Elsevier, New York.

Wilensky, R. 1981. Meta-planning: Representing and using knowledge about planning in problem solving and natural language understanding. *Cognitive Science*, Vol. 5, No. 3. 1981.

Wilensky, R. 1982. Talking to UNIX in English: An Overview of a UC. In the *Proceedings of the National Conference on Artificial Intelligence*. Pittsburgh, PA. August, 1982.