

# Using Plans to Automate Software Applications

**Jon R. Wright**  
Shannon Laboratory  
AT&T Labs - Research  
Florham Park, NJ 07932

## Abstract

Many software applications consist of a number of interdependent steps and are executed under the supervision of a human administrator or operator. The administrator responds to error conditions and checks to see that the dependencies between steps has been satisfied before allowing the application to continue. AI planning systems, because they have the capability of responding to a dynamically changing environment by replanning, are an excellent candidate for automating such applications. We report on a project that uses AI planning technology as part of the solution in automating a corporate data mining application.

## Introduction

The bandwidth of modern networks, cheap computing hardware, and the availability of new software tools have had an effect on the nature of computer applications. These applications are often distributed, and may have a complex series of interdependent steps which are under the control of a programmer or application administrator. Running the application consists of issuing a series of commands on the appropriate host. Often the output or results of one step is a precondition for the next step in the sequence. Many tasks in network management have this character, and, in our experience, practical data mining applications are often structured like this.

We are interested in planning as an automation technology that can be used to simplify the human task load required to manage computer applications. Related work includes the use of planning to generate grid workflows at the Information Sciences Institute (ISI) (Blythe, Deelman, & Gil 2004). This approach implements a planning and control layer over the Globus (Ferreira *et al.* 2002) grid computing system. State descriptions such as file locations and computing resources are extracted from Globus and plan operators are used to search over a state space to find a solution.

We have been using planning to automate an established data mining application. We rely on a middleware layer to supply state descriptions, to manage resources, and to execute plan actions, and have added a control layer based on planning. Our problem domain is probably simpler than that

of grid workflow computing. But we are working with a visible and important application with a rigid recurring monthly schedule, and both we and our clients believe that there have been genuine benefits in what has been accomplished thus far.

## Application and Problem Description

We are working with a data mining application that is executed once a month on a linux cluster. The number of nodes in the cluster is relatively small as clusters go, but each node is configured for a large amount of disk space. The base data for the application is contained in large number of files, typically more than 100,000. New files are received every day, around the clock.

The middleware layer over which the application runs is called Ningai and is described in Hume and Daniels (2002). It has some very interesting attributes including replication machinery that protects against data loss, and an infrastructure that supports command execution on a large scale.

To run an analysis over a large number of files resident on a Ningai cluster, a programmer/administrator writes a program in the Ningai command language that specifies the input files, output files, and executable required for each job step. It is not at all uncommon for a program to include 7,000 or 8,000 job steps. There are also a variety of resource parameters used to manage the execution of a program and these need to be set properly for a successful run.

An individual job step can fail for any number of reasons. The most common is that one or more of the input files are not available or do not exist in a state that permits the job step to go forward. For example, jobs are not allowed to run if one of the input files is in the process of being replicated. Being *in the process* does not necessarily mean that a file is actively being replicated. It may simply be in a Ningai job queue, and can remain in that state for a long time. Actions can be taken to improve the file's position in the job queue, so it's not strictly a matter of waiting until something happens.

The Ningai clustering layer is used to extract all the needed data about files and resources. Actions are executed through the Ningai layer as well, chiefly by writing and submitting a program. Because of the scale of the data, there can be a significant lag between the time a program is submitted and the time it completes successfully. It is not

uncommon for a program to run for multiple days before completing.

If all goes well, the process can run largely unattended, but it frequently does not. If a human is in charge of running the application, that person needs to watch for certain preconditions to be fulfilled before performing the next step in the sequence. If errors occur or there are incomplete jobs, the causes of those problems must be investigated and corrected, and those job steps must be either allowed to complete or resubmitted.

Despite the considerable leverage provided by the Ningai middleware, monitoring and detail work on the part of a human administrator is required to bring the application to a successful completion. It is the ability of a planning system to respond to a changed environment through replanning and still complete successfully that attracts us to this technology.

### The Planning System

Our planner is a simple forward state-space planner that uses heuristic search with an evaluation function specifically tailored for our problem. It is written in Perl, largely because of Perl's strength in system programming and scripting of administrative functions allows the planning engine to easily interact with its environment. The planner uses plan operators to search a state-space for a solution. Additional operators are provided to establish or re-establish the pre-conditions required for individual steps when conditions change or errors occur.

The plan is represented as a triangle table (Fikes, Hart, & Nilsson 1972). The technique is old, but is a fit with our simple planner, and provides an adequate basis for plan monitoring in our application with limited, but adequate replanning. To determine the status of a plan and determine whether it is on target to achieve its goal, state information is extracted from the Ningai middleware and matched against the triangle table. The highest numbered matching kernel of the table indicates what state the plan is in and the action that should be performed next. If no matching kernel is found, the original plan is no longer valid and replanning is required.

Actions in our application can take a long time to complete, and we have augmented the triangle table with an in-progress flag to keep the plan on track while waiting for an action to complete.

Individual plan actions can include the generation and submission of a program to the Ningai clustering software. The status of that program in terms of completed and pending job steps, job steps with missing or incomplete resources, and job steps that ended in error is passed back into the planner as state description, and that is used to determine the next action, whether to replan, wait for an in-progress action to complete, or execute the next action in the plan.

### Discussion

The planner is currently in use and has been since August, 2006. It is not an experiment and does a real job that is appreciated by our clients. An important advantage of using an explicit representation such as a triangle table as the core

of an application is that it provides a solid basis for providing up-to-date and accurate status reports. The data mining results have a hard monthly deadline, and it is an advantage to be able to provide, both to ourselves and our clients, accurate status as the deadline approaches.

We understand the data mining application with which we are working quite well, and therefore it was convenient target for automation. The process consists of a series of steps in which the output or results of one step provide the pre-conditions that must hold for one or more subsequent steps to be performed successfully. Users do not receive immediate feedback when individual steps are performed. The results of performing actions are strung out in time, and, in order to determine when the next step in the process can be performed, users have to monitor the results of the action by extracting data from the environment. It is difficult to script applications that have this nature because it is hard to orchestrate when individual steps can be launched using traditional scripting methods.

This kind of structure is relatively commonplace, and, in fact, many software applications extract and combine data from various sources in preparation for some kind of analysis. Often these processes are partially automated using scripts but require a human to oversee the process to completion, primarily to respond to error conditions and mistakes.

A simple planning engine might be able to fully automate such applications and could do a better job in responding to errors because of its ability to replan. In fact, composing plan operators bears some similarity to script writing, because a good script contains a lot of information about the preconditions necessary for the script to complete successfully. It's just that in the case of a script, error messages are issued to help a human understand the preconditions that need to be restored for successful completion of the script. This makes us think that there might be a useful role for a planner to sub as a high level scripting engine where fully hands-off automation is desired. Repetitive, tedious tasks required for good network management come to mind, such as the various tasks required to enforce a good security policy over a large community of users.

### References

- Blythe, J.; Deelman, E.; and Gil, Y. 2004. Automatically composed workflows for grid environments. *IEEE Intelligent Systems* 19(4):16–23.
- Ferreira, L.; Berstis, V.; Armstrong, J.; Kendzierski, M.; Neukoetter, A.; Takagi, M.; Bing-Wo, R.; Hernandez, O.; Magowan, J.; and Bieberstein, N. 2002. Introduction to grid computing with Globus. Technical report, IBM Redbooks.
- Fikes, R.; Hart, P.; and Nilsson, N. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4):251–288.
- Hume, A., and Daniels, E. S. 2002. Ningai: A linux cluster for business. In *FREENIX Track: 2002 USENIX Annual Technical Conference*, 195–206.