A Knowledge-Based Approach to Encouraging Reuse of Simulation and Modeling Programs

Lawrence H. Miller
The Aerospace Corporation
Post Office BOX 92957
Los Angeles, CA. 90009-2957

Alex Quilici
Department of Electrical Engineering
University of Hawaii at Manoa
2540 Dole St, Holmes Hall 483
Honolulu, HI. 96822

Abstract

This paper describes a knowledge-based approach to encouraging the reuse of existing simulation and modeling programs. In our environment there are two barriers to reuse of these programs: poor interfaces and minimal documentation. To get around these problems, we are (1) treating each existing program as an operator in a planning system, (2) creating a knowledge base describing the user goals each program achieves, the pre- and post-conditions of running the program, and its I/O behavior, and (3) developing several tools that make use of this knowledge to automate the development of new interfaces to these programs and to assist the creation of scripts that achieve high-level user goals.

1 Introduction

The Aerospace Corporation has a large collection of programs that perform simulation and modeling tasks related to spacecraft design and evaluation. As with most corporations, there exists a strong desire to make as much use of existing code as possible, and to write new code only as a last resort.

Unfortunately, these programs have several properties that make this goal extremely difficult to achieve. First, they were written in Fortran by engineers and physicists untrained in software engineering methodologies. The result is poor program design, which makes it all but impossible to separate these programs into reusable components. Second, they often started out as one-shot solutions to a particular program and only over time proved more generally useful. Because these programs were originally intended to be used only by their designers, they tend to have horrible interfaces and minimal to non-existent documentation which discourages their use by anyone other than the small community of users that are currently intimately familiar with the program.

Our efforts have concentrated on ways to maximize the use of these existing programs. In particular, we're concerned with developing techniques that encourage the use of existing programs as components in the creation of new software systems. Thus, our focus is on increasing the reuse of programs as a whole rather than on their components, although that's clearly an important issue as well [Bigg 90].

Our basic approach has been to create a knowledge base that describes our existing simulation and modeling programs, and to then use this knowledge base in two ways:

1. To help construct new interfaces that can be wrapped around our existing programs.
2. To help construct scripts for achieving high-level user goals that use our existing programs as components.

Currently, we're in the process of developing a pair of tools that use this knowledge base. The first takes a description of a simulation program and automatically generates a new interface that can be wrapped around it. The second allows developers to form scripts that achieve high-level user goals by connecting up goal-oriented descriptions of the function of various programs.

2 The Knowledge Base For Simulation Programs

We view each of our simulation and modeling programs as an operator in a planning system. Thus, our knowledge base describing these programs contains information similar to that kept by most planning systems [Alle 90]. In particular, we describe the goals each program achieves, the situations in which it's applicable, and its input and output behavior (which corresponds loosely to the preconditions and postconditions of the program).

As an example, one very useful program, IMPACT, simulates a collision between two orbiting space-objects. Our knowledge base contains the following information for this program:

1. It achieves the goal "simulate collision between two space craft."
2. It is applicable under various constraints on the type of collision, such as that the collision must be orbital and hyper-velocity.
3. Its input is a file "impact.in" that contains the initial collision parameters, such as the initial velocity of the projectile in meters per second and the altitude of the collision in kilometers.
4. Its output is a file "impact.out" that contains a description of the initial orbits of the particles that result from the collision, presented as a set of particle positions and velocity, grouped by size of particle.

The information describing the various input and output values of these programs is quite detailed, including the value's purpose, its expected units, its expected range, and so on. In addition, since some of these input values can be computed automatically by running alternate programs, the knowledge base also associates input values with the programs that can be run to compute them.

We construct this knowledge base in the obvious way: first, by examining whatever program documentation happens to be available, and second, by consulting with current users of the program. Thus, in addition to its role in assisting the process of constructing new interfaces and forming scripts that use these programs, the knowledge base serves as additional documentation for these programs.

3 Providing Multi-Modal Interfaces

A key to making existing programs usable to a larger community of users is to provide them with a friendly user interface, complete with dialog boxes and mouse selections for input, and bar charts, plots, and tables for output. Unfortunately, our current fleet of aging simulation and modeling programs have interfaces that are exactly the opposite. Typically, these programs accept a stream of values for input, in a particular order and in particular units, and produce a stream of values as output. It takes a significant amount of time for users to come up to speed on the input formats and to understand the meaning of the program's output.

Because these programs are generally poorly written, it's difficult to perform cosmetic surgery on them. They frequently read input or produce output from multiple locations in the program, or read and write input and output values directly into specific locations in memory. Their batch nature, however, makes it straightforward, although time-consuming, to wrap them in a richer interface. The input side of the interface prompts for values or runs other programs to provide the needed values, and then creates the files necessary for a given program to run. The output side of the interface takes the program's output values and displays them in appropriate tabular or graphic forms.

Originally, we created these improved interfaces by hand. In particular, we built an improved interface to an aging Fortran program that computes reliability of satellite components, and included it as part of a new, multi-modal environment for evaluating satellite performance. Given this knowledge base, however, it becomes possible to automate much of this process. We've created a straightforward interpreter that's given existing programs to run, and uses the knowledge base to automatically query users or run other programs to determine the necessary input values. It then creates the necessary input file and runs the program. Once the program runs, the interpreter uses the knowledge base to run through its output file and display those output values appropriately.

By providing these programs with improved interfaces, we have greatly extended the community of users for these programs. In addition, our knowledge-based approach is much simpler than trying to construct new interfaces by hand, or trying to modify the existing simulation programs themselves.

4 Synthesizing Simulation and Modeling Scripts

Another key to encouraging reuse is to make it easy for users and developers to combine existing programs to form solutions to their problems. In our environment, one important set of programs simulates orbiting space objects. Along with IMPACT, these include programs to determine the probability of collisions with other space objects, how long particles will orbit, and where particles are likely to land when they return to earth. Experienced users of these programs have figured out how to combine them into scripts that achieve their high-level goals, such as planning satellite breakups meeting certain constraints, or determining whether an existing breakup will cause damage to a particular satellite. Most of these scripts haven't yet been implemented as programs, and exist solely in the heads of users.

Our feeling is that there are three ways to increase the reuse of our existing simulation programs. First, by capturing existing uses of programs as concrete scripts that solve particular user goals. Second, by assisting users in automatically creating new scripts. And third, by presenting these scripts in a way that they can understand better what the component programs do.

Our assumption is that the scripts these experienced users formulate can be represented as a hierarchical goal decomposition, consisting of an initial goal decomposed into subgoals, with programs that achieve various subgoals attached to leaves of the tree. As an example, consider the goal of planning a breakup collision that satisfies the constraint that there's a low probability of debris colliding with an existing satellite. It breaks down into the following steps:

1. Decide upon an initial collision velocity and collision angle.
2. Simulate the collision
3. Determine the probability of hitting an existing satellite.
4. Reposition existing satellites that are in danger.
5. Determine a new collision velocity and angle.
6. Go to step 2.

Some of these subgoals correspond directly to running programs, such as step 2, is achieved by running the IMPACT program. Others, however, correspond to another sequence of subgoals, such as step 3, which consists of a subgoal of determining which satellites are in the general orbit of the particles resulting from the breakup, and then determining the probabilities of each of these satellites being in danger. Finally, some subgoals correspond to manual tasks, such as steps 4 and 5, which involve manually trying to find new locations for satellites in danger and trying to guess more appropriate collision angles.

Our approach is to provide a graphically-oriented tool that allows users to specify the complete goal hierarchies for solving their problems, along with the connections between various subgoals and existing programs. This allows them to easily form a library of scripts that capture how the programs are actually being used to solve problems. We then provide an interpreter that automatically executes these scripts, automatically obtaining needed input values and displaying output values appropriately. In addition, while executing these scripts, the interpreter highlights active subgoals, allowing users to learn the uses of existing programs.

Our knowledge base supports this process by containing the knowledge necessary to map programs to the goals they achieve, as well as knowledge about their inputs and outputs, so that users constructing scripts are freed from worrying about those values.

5 Future Work and Conclusions

Currently, there are several limitations to our approach. First, our programs are primarily batch and have limited options. We haven't yet tried to create a knowledge base for an interactive program or for a program that does different things in different situations. And second, our approach is only at the prototype stage. We have implemented knowledge bases describing a set of approximately a dozen programs and scripts of their use, as well as first pass implementations of the tools that use this knowledge base to provide new interfaces, and to assist users in entering and executing the scripts for their high-level goals. We are beginning to consider many of the issues that will arise when we scale up to larger collections of programs, such as users being able to find the appropriate script that achieves their goal [Fisc 91].

Besides beginning to deal with scaling issues, we're also starting to explore several new directions. The first is to use the knowledge base to automatically form scripts from initial descriptions of user goals. In particular, we've been considering how to combine our approach with *Weaving*, a large grained data flow environment designed to support the combination of programs [Gorl 91]. Another problem we've begun looking at is to tryito automatically extract some of the information now placed into our knowledge base by hand. In particular, we're focusing on how we can extract knowledge about a program's input and output behavior.

In summary, we are trying to encourage the use of these batch programs by making them appear to be interactive and by including them as components of more sophisticated interactive, software systems. We're doing so by (1) providing them with better interfaces and by (2) combining them into high-level scripts that reflect how they're used. The key is we're using a knowledge base describing these programs to help us create tools that automatically perform these tasks.

References

[Alle 90] Allen, J., Hendler, J. and Tate, A. (Editors), *Readings in Planning*, Morgan Kaufman, Palo Alto, CA, 1990.

[Bigg 90] Biggerstaff, T. (Editor), *Software Reuse*, Addison Wesley, Reading, MA, 1990.

[Fisc 91] Fischer, G., Henninger, S., and Redmiles, D., "Cognitive Tools for Locating and Comprehending Software Objects for Reuse." In *Proceedings of the 13th International Conference on Software Engineering*, IEEE Computer Society Press, 1991.

[Gorl 91] Gorlick, M. and Razouk, R., "Using Weaves For Software Construction and Analysis." In *Proceedings of the 13th International Conference on Software Engineering*, IEEE Computer Society Press, 1991.