Modernizing Fortran 77 Legacy Codes

The investment in established codes is preserved as modern capabilities are added.

NASA's Jet Propulsion Laboratory, Pasadena, California

An incremental approach to modernization of scientific software written in the Fortran 77 computing language has been developed. This approach makes it possible to preserve the investment in legacy Fortran software while augmenting the software with modern capabilities to satisfy expanded requirements. This approach could be advantageous (1) in situations in which major rewriting of application programs is undesirable or impossible, or (2) as a means of transition to major rewriting.

Programs written in Fortran 77 and other early versions of Fortran retain much intellectual and commercial value. These codes have been carefully validated and often perform excellently, even on modern computers. However, the early versions of Fortran are often not adequate for the computations needed for the increasingly complex systems typically encountered in current practice. For example, early Fortran programs may not utilize dynamic memory or their data structures may not reflect problem domains well. Often, user interfaces are poor or nonexistent. On the other hand, modern programming languages (most notably, object-oriented languages) offer much better support for complex programming projects.

The developers of the present incremental approach have found that contrary to the conventional wisdom among object-oriented programmers, it is not necessary to redesign codes from the bottom up to make them object-oriented. Many object-oriented programs make use of non-object-oriented libraries for basic functions — for example, calling operating systems. The important question in modernization of a legacy code is whether the subroutines in the code have sufficient basic functionality. For most legacy codes that have been in use for a decade or more, this is the case. The weaknesses of legacy codes arise with respect to flexibility, extensibility, and related issues.

The present incremental approach is based partly on the assumption that such weaknesses can be addressed at a higher level, by building interface software libraries that mediate between main programs and underlying legacy codes. Major goals in this approach are to increase program safety, simplify interfaces to subroutines, add dynamic memory management, encapsulate different parts of programs, add abstract data types that reflect the problem domains, and add or improve user interfaces.

The present incremental approach can be characterized as one of building modern superstructures around legacy codes rather than completely rewriting those codes. Building such a superstructure increases the opportunity to reuse the intellectual capital already invested in the legacy code and entails less risk that the rewrite might not work properly. As much as possible, one avoids modification of original subroutines; instead, one strives to embed the modern features in an interface library.

This approach admits of several methodologies. One such methodology is based on Fortran 90, because this language has modern features yet maintains compatibility with older Fortran software. Other methodologies could be based on other modern computing languages (for example, C++) in which one can write programs capable of calling the original Fortran subroutines. Once software constructed following this approach works correctly, there is always the option of replacing individual pieces of the legacy code, and eventually even the entire code. One important advantage of the use of a software superstructure is that the legacy code can still be used during a modernization effort.

This work was done by Viktor Decyk and Charles Norton of Caltech for **NASA's Jet Propulsion Laboratory**. *Further information is contained in a TSP (see page 1).*

This software is available for commercial licensing. Please contact Don Hart of the California Institute of Technology at (818) 393-3425. Refer to NPO-21166.

Active State Model for Autonomous Systems

Autonomous systems would be able to diagnose themselves and respond accordingly.

NASA's Jet Propulsion Laboratory, Pasadena, California

The concept of the active state model (ASM) is an architecture for the development of advanced integrated fault-detection-and-isolation (FDI) systems for robotic land vehicles, pilotless aircraft, exploratory spacecraft, or other complex engineering systems that will be capable of autonomous operation. An FDI system based on the ASM concept would not only provide traditional diagnostic capabilities, but also integrate the FDI system under a unified framework and provide mechanism for sharing of information between FDI subsystems to fully assess the overall "health" of the system.

The ASM concept begins with definitions borrowed from psychology, wherein a system is regarded as active when it possesses self-image, self-awareness, and an ability to make decisions itself, such that it is able to perform purposeful motions and other transitions with some degree of autonomy from the environment. For an engineering system, self-image would manifest itself as the ability to determine nominal values of sensor data by use of a mathematical model of itself, and self-awareness would manifest itself as the ability to relate sensor data to their nominal values. The ASM for such a system