CASHIN79

Cashin, P.M., Joliat, M.L., Kamel, R.F., and Lasker, D.M.; Experience with a Modular Typed Language: PROTEL; Proceedings of the Fifth International Conference on Software Engineering 1979.

Summary of Ideas

The authors conclude from their experience using PROTEL in large systems development projects since 1975 that modularity with strict type enforcement across module boundaries is necessary to keep the projects manageable and maintainable.

Modules should separate their interface sections from their implementation sections; multiple interface sections can be useful; module name qualification of exported procedures is needed in any large set of modules, and there is a need to identify sets of modules as a whole.

The PROTEL solution to installing and using optional modules in production software by binding to procedure variables works well. Type transitivity from indirectly imported modules is good overall but there are implementation efficiency problems with this.

Any large software project needs a configuration control system as well as a system to recompile only modules that are actually affected by a change in another module.

Important Points

Separation of interface and implementation sections allow people using the module to simply look at the interface and know that the implementation will work. This also localizes the effects of implementation errors and the necessary recompilation, as well as allowing for several different implementations of the same interface, which is good for system configuration.

Multiple interface sections are good because they allow the separation of module functions by the user (for example general and privileged functions in a file system). Multiple implementation sections are good at minimizing recompilation and for imposing internal structure on a module.

A simple linear chain of interface sections and a single implementation, however, may be sufficient and more efficient. PROTEL uses an inverted tree structure of interface and implementation sections.

Large software projects require the identification of sets of modules as subsystems (called AREAS in PROTEL), this can be thought of as a configurable package of features.

Optional software in a large system should not be done through conditional compilation due to inefficiencies but rather by procedure variables which are set when the optional module is loaded. This was found to be very useful in active systems which cannot be brought down without difficulty (such as telephone switching computers).

Type transitivity from other modules causes a problem in determining the scope of an interface type change, but overall this feature was found to be worthwhile.

Because interfaces are easy to change in PROTEL, it was done often, causing much recompilation. This required more sophisticated systems to handle recompilation of only those parts that were actually affected by the changes.

Type checking in PROTEL is done by imbedding interface information in object modules. The compiler then takes this information from each module that is visible in a module being compiled and preloads the symbol table. This was found to be inefficient and a new system was developed to only imbed symbols actually required (not just visible but used) and to keep symbol table definitions already loaded for multiple implementation sections.

Relevance

This article summarizes actual experience that gives several justifications for the current belief that languages to be used in large software development projects need to have module mechanisms that provide type checking across module boundaries and that have explicit interfaces.