Tichy's Response to R. W. Schwanke and G. E. Kaiser's "Smarter Recompilation"

Schwanke and Kaiser's extension of smart recompilation is an intriguing idea. Their mechanism aims at delaying recompilation work by permitting "harmless" compilation inconsistencies to remain after changes. Full consistency can be reestablished at a later time, after the change has been tested in a subpart of the system. If the change was inadequate, then no needless compilation work was performed. This strategy is used frequently in practice, by exploiting loopholes in system generation tools. Schwanke and Kaiser's mechanism is novel in that it makes this practice safe. The compiler is aware of the inconsistencies, and will not overlook dangerous ones. Furthermore, it can help reestablish full consistency once a change is deemed acceptable.

Smarter recompilation defines a harmless inconsistency as follows. If a declaration is changed, this action is treated as introducing a new version of the declaration. The coexistence of both the old and new versions in the same configuration is a harmless inconsistency as long as the uses of the two versions do not conflict. The system must be separable into two partitions, one that uses the old version and the other the new one, such that the interface between the two depends on neither. Since the inconsistent declarations do not cross the interface, the two partitions may even communicate with each other. Of course, inconsistencies not captured by the type system cannot be treated in this way.

Schwanke and Kaiser's note leaves a few minor questions unanswered. For instance, a new or changed declaration might cause a redeclaration or overloading error that can only be detected by recompilation. Is this potential problem left undetected until full consistency is desired, or is it checked immediately? If a declaration is deleted that is still in use, is the deletion treated as an error or as a *delayed* deletion that will take effect after the last use disappears? If the old and new versions of a procedure operate on the same data structure, is it always desirable to let both versions coexist, or can the programmer indicate that the old version should be eliminated before the next program execution? Perhaps a future paper about an implementation will clarify these points.

Smarter recompilation also opens the door for more powerful programming tools. For example, since the mechanism maintains cross-reference information, a tool like Masterscope [1] could be built relatively easily. The tool would have the advantage that cross-reference information is immediately available once a module has been compiled. Only little additional data would be needed to classify the uses of symbols. The information could also be exploited by a Maintainer's Assistant. This program helps with reestablishing consistency after changes by suggesting corrections of the affected program parts. For example, it could attempt to make call sites of changed procedures consistent with their headers,

update operations on changed record fields, or perform some simple program transformations in response to data structure changes.

WALTER F. TICHY
Informatik II
University of Karlsruhe
D-7500 Karlsruhe 1
FR Germany

REFERENCE

1. KAISLER, S. H., *Interlisp, The Language and Its Usage.* John Wiley, New York, 1986.