Date: 8 December 1997
To: J3
From: Van Snyder
Subject: Enhancing Modules III - Miscellaneous

There are several module-related issues that have not yet (or not recently) been the topic of a paper. These issues are presented here only to keep awareness of them. No concrete solutions are offered.

- `PRIVATE` and `PUBLIC` refer to names, not to entities. Thus it is not possible separately to control the visibility of specific and generic procedures, or of several different generic interfaces, of the same name.

- Module procedures have specific, not generic interface. Accessing them by use association allows greater possibility of conflict with another procedure of the same name than would be the case if they were defined to have generic interfaces.

- Addition of an annotation of a `USE` statement to indicate that resources used from the module shall be qualified by the module name would reduce the need for renaming clauses. In addition, it would allow one to indicate, at the points of usage, the module from which an entity is USE'ed. E.g. if one has `USE POINTS, QUALIFIED`, declaring an object of type `POINT` would require using `POINTS%POINT`. If the `QUALIFIED` annotation were absent from the `USE` statement, one could choose whether to use module entities with or without qualification. E.g. `POINTS%POINT` would be allowed, but not required.

- It is frequently useful to allow users of a module or type to reference data accessed from the module, or components of the type, but not to change them. The present mechanism for this functionality is to provide a public procedure to view a private datum, but no public procedure to change it. These procedures are usually trivial. One hopes that "Quality of implementation of compilers" includes materializing these procedures in-line, but it usually doesn't. Another drawback is that the lifetime cost of a program is roughly proportional to its bulk. Requiring gratuitous procedure-ification of this functionality increases the bulk of programs, and therefore their cost.

  Providing an un-symmetrical visibility attribute, spelled `READONLY` in some proposals, would allow more efficient programs, even in "low quality" environments that don't in-line trivial procedures, and reduce code bulk, thereby presumably reducing programs' lifetime costs.