Automatic export of identifiers from the definition module

Arthur Sale

Professor of Information Science - University of Tasmania
GPO Box 252C Hobart Tasmania Australia 7001

The BSI Modula-2 Working Group has suggested that Professor Wirth's proposal to discard the export list of a definition module be rejected (Cornelius, 1986). In their report to the Modula-2 Users Association the Working Group state

'[WG084] M2WG has agreed to retain the original syntax and semantics, ie objects which are to be exported from a definition module have to be listed in an export list.'

This paper argues that this suggestion should not be adopted and that the 1984 revision to automatically export all identifiers declared in a definition module should be retained.

ANALYSIS

Several correspondents to the MODUS Quarterly (November 86) have pointed out that few if any definition modules export identifiers selectively. This provides a strong *prima facie* case for assuming that the export clause in a definition module is redundant. There is also a strong case on theoretical and consistency grounds for rejecting selective export. However a more careful analysis of the situation is required to ensure that some unusual but important facility is not being overlooked by deleting it. Accordingly this paper will systematically examine the arguments for and against selective export. In the ensuing discussion the word *visible* (and its derivatives) will be meant to refer to an item being readable in the text of the definition module, and the term *accessible* to refer to an ability to refer to the item by imported identifier in a using module.

Firstly, we should ask when an identifier *must* appear in a definition module. Assume that the valid reasons for an identifier appearing in a definition module are based on it it serving a syntactic purpose in either a calling module or the definition module. The following syntactic purposes can be identified:

- the identifier will be exported, or
- the identifier is required for a subsequent using occurrence in the definition module, or
- the identifier is required as a by-product of one of the other two requirements.

Secondly, would anyone wish to transfer the defining occurrence of an identifier from an implementation module to the corresponding definition module? The only plausible reason for such a practice would be that the writer of the module wishes to not publish the text of the implementation module *and* publish the structure or definition of some object which is part of the implementation. This practice should not be encouraged. The definition module should constitute the module writer's contract with the user, and extraneous material should not be included in it.

Selective export allows the module writer to Include such defining occurrences in the definition module and yet not export them; the deletion of the export clause would remove this possibility. However this argument for its retention is weak and a similar purpose can be achieved, if absolutely necessary, by either publishing the text of the implementation module or by including a suitable comment in the definition module. This point is crucial, because it will be shown that there is no other plausible use for selective export.

VALID DEFINING OCCURRENCES

Procedures

A definition module may only contain a procedure heading, not its body. While the heading may contain a parameter list, the parameter identifiers have no significance to the user of a module and are never exported. Their only significance, if any, is for checking against the (redundant) heading in the implementation module. The only components of a procedure which are exported are its identifier and the structure of its parameter list.

Since no bodies of procedures can occur in a definition module, and the definition module has no initialization section of its own, there can be no using occurrences of the procedure identifier (calls or activations). This means that a procedure can never be required as the consequence of something else--4t can only be a valid component of a definition module if it is to be exported.

Variables

It has been argued in many places that modules should not export variables as they represent severe security risks to the integrity of the module's correctness. For example see Sale (1986b). However, if a variable does occur in a definition module, there can also be no using occurrences (references) to it, for precisely the same reason as for procedures. Again a variable can never be required as the consequence of something else-it can only be a valid component of a definition module if it is to be exported.

Constants

Now consider the case of constants declared in a CONST part. These may be intended for export only, or may be referenced in subrange type declarations in the definition module. The first case is relatively rare in practice but can occur as defining the maximum size of some resource, or in providing identifiers for common constant values such as the IS0 character set. Other examples are given in Sale (1986a). Not exporting an identifier intended for export is senseless.

The second case is much more common, yet it still does not offer an argument for selective export, for if the constant defines one of the limits of a user-accessible subrange type, then the limit values are valid information for the user and may be useful in **FOR statements (for** example). In any case it would be pointless to try to hide them because the accessibility of the type enables their values to be retrieved by the MIN or MAX functions, or failing that by the appropriate type transfer function (inverse to ORD). The only facility not available to a module user who is given exported access to a subrange type but not its limits is the facility to use (syntactic) constant expressions involving the limits, thus prohibiting the declaration of derived constants or subrange types. Even this would be possible if the proposal to allow standard function calls (eg MN, MAX) in constant expressions is implemented.

Types

Since no other kind of object requires selective export, any case for it must lie in the area of type declarations. One possibility is for an exported type to be defined in the TYPE part of a definition module, but for any named identifiers in its internal structure to be not exported. This is a sort of opaque export: the structure is visible but cannot be used. Let us examine the possibilities:

Type synonyms
Declarations of the kind

```
        Identifier1 = Identifier2
```

are relatively rare in definition modules, but introduce no new issue. Since the type is simply given a synonymous name, the purpose of this is probably to export it, but in any case whether selective export is important or not depends on whether the type Identifier2 is accessible anyway.

Subrange types

If the host type is accessible to the user, then not exporting a visible subrange type is almost pointless. Type identity in variable parameter compatibility is the only case where similarly declared subrange types are distinguished.

Enumeration types

A new issue is introduced with enumeration types. Does it make sense to export an enumeration type identifier but none of the associated constant identifiers? Or to export only some of the associated constant identifiers? The first question is almost equivalent to exporting the identifier opaquely, and is discussed later. The second implies that there are values which the user can see but not access by identifier. Since all values of a visible enumeration type can be reconstructed anyway by type transfer (given that they are visible), this seems pointless. The *Modula-2 Report* (Wirth, 1982) and most Modula-2 compilers with explicit export resolve this issue by simply not allowing selective export of enumeration types: if the type identifier is exported, so are all the constant identifiers which may not themselves be explicitly exported.

Record types

Of all the structured types, only record types involve the defining occurrences of internal identifiers: the field identifiers. These identifiers are not in the scope of the definition module and correspondingly are not in the scope of an export clause. There is only one sensible approach to the export of field names and this is stated in the *Modula-2 Report:* they are exported associated with the type identifier.

Set types, array types and pointer types

These types involve no internal defining occurrences.

Procedural types

Procedural types involve no internal defining occurrences. In any case, the visibility of a procedural type declaration (regardless of export) allows a user to reconstruct compatible types and procedures, since compatibility is determined by structural rules, not type identity. There is no reason at all for not exporting a visible procedural type.

Opaque export

Many implementations of Modula-2 restrict opaque export to types which are subsequently declared to be pointer types. This restriction is a compiler convenience, as it permits simple implementations. However, even with it, implementation modules can be written which provide any desired type-all that is necessary is for the pointer " to have the desired type as its bound type. It should also be pointed out that there are compilation techniques for entirely removing this restriction. A quality Modula-2 implementation could permit an opaquely exported type to be declared in the implementation ~ule with any type.

SUMMARY

The valid inclusion of a variable or procedure in the definition module implies its export. (If this is not the case then the definition module is overspecified and the non-exported object should be transferred to the implementation module.) The selective export of constant identifiers has been shown to have no useful purpose. The only possible case for selective export arises with type declarations. This is focused on one issue: the export of a type identifier whose defining occurrence occurs in the definition module but the non-export of any internal identifiers in its structure. The user really wants an opaque export of a structured type and attempts to achieve this by having the structure visible but not exported. This is a bad response to a poor situation: the proper solution is to encourage Modula-2 processors to implement opaque export so that the details of any type in the implementation module can be opaquely exported. There are techniques for doing this. Alternatively the existing opaque export of a pointer type can be used to provide the desired facility. The conclusions are simple:

1 Deletion of the export clause and automatic export of all identifiers whose defining occurrence occurs in the definition module is an improvement in and a simplification of the language.

2 Why should local *modules* retain their own idiosyncratic structure? Should they not also have interface and implementation components syntactically parallel to separate modules?

REFERENCES

CORNELIUS, B. (1986). ‘Significant changes to the language Modula-2’. *MODUS* Quarterly, 6, pp8-14.
SALE, A H J. (1986a). *Modula-2: Discipline & Design.* Addison Wesley.
SALE, A H J. (1986b). 'Improving the quality of definition modules.' *MODUS* Quarterly, 6, pp27-29.
SALE, A H J. (1987). 'Optimization across module boundaries.' Aust. Comp. Jnl. (to be published 1987).
WIRTH, N. (1982). *Programming in Modula-2.* Springer-Verlag.