In this paper some views are presented on the way in which complex systems, such as Operating Systems and the programs to be interfaced with them can be constructed, and how such systems may become heavily library oriented. Although such systems have a dynamic nature, all interfacing within and among modules can be checked statically. It will be shown that the concepts presented are equally valid for single user systems, multi-programming systems and even distributed systems. The ideas have been spurred by the implementation of a modular version of Pascal and a supporting Operating System, currently nearing completion at Twente University of Technology, The Netherlands.
1. Co-operating Modules

1.1 Modules

The basic programming tool we consider in this paper is a module. This concept is incorporated in several recent languages (ADA[1] (where it is called "package"), Concurrent Pascal[6] (where it is called "class"), Pascal Plus[17], LIS[13], MESA[15], Modula[19], Modula-2[20] and many others). Although details may differ from one language to another, the following description should suffice for the sake of this paper:

A module is a set of related (type-)definitions, data declarations, operation declarations (viz. procedures and/or functions) and a section describing the initialization (sometimes also the finalization, as in Pascal Plus) of the module's local data.

In order that programs (or systems) may be composed from co-operating modules some of the declarations within modules may have to be made accessible outside these modules. We will say that these declarations are exported from these modules. Conversely, the use of items declared in other modules will be called "importing". If a program is composed from several modules, then the rules according to which modules may be interconnected determine the accessibility and scope of the objects within the program. It needs no arguing that the visibility structure that can be obtained in this way may well differ from the scope rules one encounters in classical, block structured languages (of which Pascal may be considered an example). A judicious use of the interconnection rules between modules may lead to a simple, but, nevertheless very powerful means of structuring systems, and in particular: Operating Systems. This we hope to show in the sequel.

1.2 Interdependency of Modules

In the following we postulate that each module specifies from which other modules it wishes to import items. Thus a program (composed from modules) may be modelled by a directed graph, where the modules are the vertices and the relation "imports from" determines the (directed) arcs of the graph. (Classical block-structure would limit the structure of such graphs to trees.)

If knowledge of the total set of modules is only used when the program is composed from a set of object modules, as is the case in traditional systems with "independent compilation", it can hardly be checked that the arguments supplied for a procedure