C. Irby

INTRODUCTION TO THE MODULAR PROGRAMMING SYSTEM

12 MAY 72

MPS 2.0

James G. Mitchell*

Xerox Palo Alto Research Center*
3180 Porter Drive
Palo Alto, CA 94304
(415) 493-1600

Stanford Research Institute
333 Ravenswood Avenue
Menlo Park, CA 94025
(415) 326-6200

1d

GOALS 2

The Modular Programming System (MPS) is a set of tools for the development and continued evolution of large software systems in an interactive environment. All such large software systems share certain characteristics: 2a

(a1) they are the work of a group of people, whose membership will change over time; 2a1

(a2) they are necessarily constructed from a number of separately developed programs; 2a2

(a3) they evolve and grow throughout their lifetimes (and there is evidence that they also "age" [Lehman & Belady]). 2a3

The MPS project aims to decrease the effort required to build and evolve such systems and to increase the reliability of the resultant products. As a specific test of its capabilities, MPS will be used in the rewriting and restructuring of the NLS system developed at Stanford Research Institute. 2b

APOLOGIA 3

Points a1, a2, a3 are axiomatic statements about the dynamics of all large software systems. The following discussion uses these and a few other axioms to establish desirable characteristics for MPS. They are intended only to lend plausibility to the set of capabilities which the MPS project is investigating. Furthermore, the "logical conclusions" only represent design choices to satisfy the axioms; other choices could certainly be made which would not be inconsistent with the axiom set, but that is another research project. Hopefully there is a minimum of hidden meaning in the following discussion: each axiom and consequence is intended to be taken strictly at face value. 3a

We first add two more axioms to the above set: 3b

(a4) Large software systems must be able to take advantage of available hardware for efficiency. 3b1

(a5) Program bugs are not known before they occur. 3b2

(a4a) a1-a4 imply that software components, hereafter called modules, should be separately compilable and debuggable. Therefore there must be a way of linking or binding separate components together to provide an environment (data and programs) within which a module can be debugged. 3c

(a6) In an interactive programming environment, users must be able to develop and use debugging tools applicable to programs

in the same programming system [Krutar] [Mitchell] [Perlis] [Teitelman]. 3d

a4a, a5, and a6 then imply that 3e

(a6a) the environment of a program must be dynamically alterable; 3e1

(a6b) a program should not have to be altered when its environment changes in ways which do not affect the semantic intent of the program [Dennis] -- this is called programming generality. 3e2

(a3a) a3 suggests that a desirable characteristic for tools for building large systems should be that the energy to change part of the system should be more a function of the complexity of the change than of the size of the system. 3f

(a3b) A new system always has parts which are functionally similar to previously developed systems. The new system may therefore be regarded as a change (though perhaps substantial) to an older system. a3a then points out the necessity for being able to reuse components which have been made reliable through usage. This increases the initial reliability of the new system and decreases its cost. 3g

(a3c) One way of constructing useful components is to build them from combinations of already existing modules (a3b). Hence there must be a way of bundling useful configurations together as seemingly atomic modules so they can be readily reused. 3h

MPS CAPABILITIES 4

To satisfy these objectives, MPS has concentrated on providing the following capabilities: 4a

control mechanisms which enable modules to be linked together with a minimum of built in assumptions about how each interprets control transfer over the link between them. 4a1

simple function call and return mechanisms alone do not satisfy this requirement. 4a1a

Data definition facilities that: 4a2

clarify the specification of the data structures which, together with control, completely specify the interfaces between modules; 4a2a

are potentially economical in space and accessing speed without being dependent on a particular machine; 4a2b

are an aid in developing and describing program components and the structure of algorithms. 4a2c

Facilities for dynamically binding the virtual objects required by a module for execution to real objects: 4a3

e.g., for binding a procedure call to a real procedure, a "typed" pointer to a data structure of the correct type, etc. The set of bindings for a module's virtual objects at a given moment comprises the environment for that module. 4a3a

Complete accessibility to the MPS "virtual machine" (which is a set of primitive MPS programs) and to MPS programs as data structures. 4a4

This enables debugging and measuring tools to be built as standard MPS programs and along with dynamic binding allows such tools to be brought to bear on MPS programs whenever necessary. 4a4a

The ability to bundle a configuration of data and program modules together as a module which may be saved for later use just as a simple, atomic module: 4a5

this allows systems to be partly initialized by partially executing them and then bundling them up for later use with the initialization computations factored out; 4a5a

it also allows a configuration which has exhibited a bug to be saved away for later perusal with the state as it was when the bug was discovered; 4a5b

lastly, it allows standard modules to be built by configuring them from other modules in the spirit of using already available components whenever possible and provides some logical completeness to the system. 4a5c