POD-HOS INTERFACES:
AN EXAMINATION OF AXES/POD RELATIONSHIPS AND OTHER ISSUES

CONTRACT N00039-81-C-0183

BGS SYSTEMS, INC.
WALTHAM, MA 02254

AND

HIGHER ORDER SOFTWARE, INC.
CAMBRIDGE, MA 02139

DECEMBER 1982

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A128301	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) POD-HOS Interfaces		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER RPT-POD-1982-5
7. AUTHOR(s) BGS Systems, Inc. High Order Software, Inc.		8. CONTRACT OR GRANT NUMBER(s) N-00039-81-C-0183
9. PERFORMING ORGANIZATION NAME AND ADDRESS High Order Software 955 Mass Ave. Cambridge, MA 02139 BGS Systems, Inc. University Office Park Waltham, MA 02254		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ACRN: Item 0004
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Electronic Systems Command Washington, DC 20630		12. REPORT DATE
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document describes an approach to interfacing BGS Systems, Inc.'s POD system and High Order Software, Inc.'s USE.IT system. It incorporates a technical overview prepared by BGS Systems and a more detailed report prepared by HOS.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

1 OVERVIEW

This report addresses the feasibility of and requirements for interfacing two powerful software engineering tools - POD (Performance Oriented Design) developed by BGS Systems Inc. and HOS developed by High Order Software, Inc. of Cambridge, MA.

POD is a software engineering tool for the life cycle management of system performance. It is a tool which provides a structured machine readable format (the System Description File or SDF) for representing a system's hardware architecture, software structure and external load demand. Based on this representation, POD provides a vehicle to calculate system performance values such as response time, throughput and device utilization.

Higher Order Software (HOS) is a methodology for defining systems in a hardware and language independent manner. It consists of a specification language AXES, which provides mechanisms for defining functions, control structures, and data types, a User System Evaluation and Integration Tool (USE.IT), which is a family of tools for defining systems using the AXES language, an Analyzer for analyzing the logical correctness of systems defined in AXES, and a family of

RATs (Resource Allocation Tools), which translate descriptions from the AXES specification language into target languages such as Fortran and (potentially) POD. The AXES language consists of both a specification language (for describing the application or other system to be built) and a metalanguage for describing new data types and their associated axioms. The latter is a facility for extending the AXES language to incorporate new types of information.

In considering the feasibility of and requirements for interfacing POD and HOS, HDS is assessed in terms of its ability to describe the required POD input specifications. In addition, consideration is given to possible HOS extensions to support POD, and an approach for interfacing these tools is suggested.

The HOS report is organized as follows: Page one of the accompanying HOS report contains the introduction; pages 3 and 4 contain a summary of POD facilities; pages 5 and 6 give an overview of HOS and USE.IT; pages 7 thru 35 describe the HOS - POD interface, which is the meat of the report; page 37 has a list of references to HOS papers (plus a reference to the POD reference manual); Appendix A (which is separately numbered) consists of pages 177 to 188 of the POD reference manual.

The report contains a description of parts of the POD SDF language in terms of AXES axioms and an example of how some

information in a typical POD SDF could be expressed using AXES constructs. It is, thus, useful and illustrative of how a complete specification of POD information in the HOS environment could be done.

The rationale for this task was to enable users to present a unified description of a system in the HOS requirement specification language (AXES) in such a way that sufficient information would be embodied in such a description so that both a higher level language (e.g. Fortran) program and a POD model could be generated from that description. In order to do this, two things must be done.

1. Conventions must be established so that the necessary performance related information can be embedded in a system specification written in the AXES language.
2. The POD SDF language must be defined to HOS so that an actual translation can be done.

Once this is done a translator (called a RAT in HOS terminology) can be built that uses information embedded in a HOS description of a system and information about the information layout used by POD to generate a POD model. HOS also has a facility to do limited consistency checking of this performance related information.

The HOS report presents two main pieces of new information:

1. A definition of a subset of the POD SDF syntax in the AXES metalanguage.

-- An example of how sample pieces of a POD SDF model of a system could be expressed in the AXES language.

The AXES metalanguage provides a facility for extending the AXES syntax by introducing new primitives that support POD concepts such as workload and device definitions. These are not normally part of a requirements or HLL specification for a system. The extension syntax is defined in the AXES metalanguage as axioms and data types. Values can then be associated with these new data types as part of a system requirements specification. As an example of how the definition of POD constructs can be produced, the HOS report provides AXES metalanguage definitions of four POD syntax constructs:

1. device type (pp.7-12)
2. file catalog (pp.12-13)
3. workload (pp.14-16)
4. workload type (p.17)

The definitions are typical of what would be required for a complete definition of the POD input language.

The HOS report also specifies a set of conventions for how information needed by POD can be expressed in the AXES language and how these specifications correspond to the information in a POD SDF. The report gives examples of how six such types of information (needed to build a POD model of a system) can be incorporated in an AXES language specification. The cc cts illustrated are:

1. loop statement (pp.14, 18-20, and 22)
2. case statement (pp.20-21 and 23-24)
3. file catalog (pp.25-26)
4. device description (pp.25 and 27-29)
5. device classes (pp.25 and 29)
6. workload specification (p.30)
7. module description (pp.25 and 31-34)

Both the HOS and POD constructs for describing each of these types of information are illustrated so that the correspondence between the two ways of describing them can be compared.

An important issue is that POD uses a probabilistic description of the differents paths through a program (choices in a TEST/CASE statement) while a conventional programming language uses a specification of the conditions that will cause one or another path to be taken. These probabilities must be incorporated into the HOS AXES requirement specification in order for a detailed POD model of an application to be built. POD allows the user to specify program path lengths, the number of times a loop will be executed, and flow of control in general as a function of data input frequencies (probabilities) that are meaningful to an end user. One conclusion of this study is that this type of information must be included in a system specification for a detailed and accurate POD model to be produced. Once this information is available, POD allows some of the parameters to be

varied interactively to analyze how changes in data input frequencies will affect system performance. Incorporating this additional information in an AXES specification should not be particularly difficult and would be an appropriate extension of the research reported in the HOS report.

The complete specification of the POD syntax as AXES axioms and conventions for embedding performance related information in an AXES requirements specification is left for a follow on task. Some of the main issues that could be addressed there are:

1. shared domains
2. parameter passing between modules and module flow of control
3. data dependence of module performance and parameterization to allow analysis of the effects of this dependence in a way understandable to end users
4. POD sources
5. incorporating POD semantic relationships in the HOS axiom descriptions.

HIGHER ORDER SOFTWARE, INC.
806 MASSACHUSETTS AVENUE
CAMBRIDGE, MA 02139

POD-HOS INTERFACES

SEPTEMBER 1982

PREPARED FOR
3GS SYSTEMS, INC.

WALTHAM, MA 02254

TABLE OF CONTENTS

INTRODUCTION

In this report we describe two related interfaces between Higher Order Software (HOS) methodology and the Performance Oriented Design (POD) system. One interface, which is detailed, is the relation between POD constructs and HOS mechanisms. Some selected POD constructs are characterized in HOS terms, and then, from part of an example of a system described in terms of POD, a corresponding HOS specification of that system is described. In this way general connections between the two systems become apparent; for example, attributes in POD become primitive operations on data types in HOS. Furthermore, this exercise suggests ways in which POD descriptions might be enhanced by using HOS. For instance, it is seen that formalization in terms of HOS makes explicit attributes that are only implicit in POD.

The second interface, which is discussed more briefly, is the concept of a POD Resource Allocation Tool (POD RAT), that is, a tool that takes a correct HOS system specification as input and automatically generates a POD description. Since HOS specifications are guaranteed to possess certain crucial properties (consistency, logical completeness, and interface correctness, for example), a POD RAT, which would perserve these properties, would ensure that the resulting POD description would also have these properties. Furthermore, a single HOS specification, after being run through a POD RAT, could equally well be run through a FORTRAN, PASCAL, or other RAT for implementation, thereby doing away with the need for further coding of the system.

Included in this report is a POD overview, an HOS and USE.IT (FORTRAN/PASCAL RAT plus other HOS tools), overview, and a discussion of these POD-HOS interfaces.

1. POD OVERVIEW

Performance Oriented Design (POD) system is an interactive facility that can be used to analyze performance related problems that arise during the design, implementation, and evolutionary development of computer based systems. It provides the following facilities to the user:

- A format (System Description File) for expressing a system design's performance characteristics including hardware and its interconnections, software, and workloads to be processed.

- A command to read System Description Files and perform certain syntax checks. (For example, invalid, redundant, or omitted descriptors are detected.)

- Commands for transforming device usage estimates from symbolic machine-independent terms to actual times.

- Commands to build and evaluate analytic queueing network models of prospective system designs.

- Commands to express the model behavior in terms of response time, throughput, device utilization, queue lengths, and other derived results.

- Commands to modify design performance parameters interactively and evaluate new designs on-line [1].

Modeling of a hardware/software system can be done on two levels using POD. On one level software structures (e.g., call structures

and resource requirements) and device capabilities (e.g., device storage and processing capabilities) are specified. In addition, workloads (a series of jobs arriving at a computer system from an external source) and their arrival information must be specified. On another level, the interaction of workloads is examined. In other words, the contention for specific (hardware) resources is analyzed.

2. HOS & USE.IT OVERVIEW

Higher Order Software (HOS) is a methodology for defining systems in a hardware and language independent manner. Systems so defined are guaranteed to be consistent and logically complete.

AXES is a specification language which is based on HOS. It provides the mechanisms to define functions, control structures, and data types. A system is viewed as a single function which is decomposed into successive levels of detail in terms of other functions. Control structures state the relationship between the original function and the functions which make up its decomposition. There are three primitive control structures: JOIN, OR and INCLUDE, which represent sequential, alternative, and parallel processes, respectively [2,3].

Objects in a system are specified using data types. Data type specifications provide the primitive operations that operate on or produce the objects of data types. These operations are primitive in that they are not decomposable, but their implementations are constrained by axioms, i.e., statements about the ways in which they can interact with each other.

Abstraction is gained by defining additional mechanisms using the primitive mechanisms or pre-defined mechanisms (and thus the primitives). These mechanisms are stored in a library and can be used where needed.

The User System Evaluation and Integration Tool (USE.IT) is a family of tools by which systems are defined (using AXES), analyzed for logical correctness (using an Analyzer), and programmed (using a Resource Allocation Tool (RAT))[4]. With USE.IT a specification is interactively constructed using AXES and checked by the Analyzer for consistency and logical completeness. The RAT then takes the correct specification and automatically produces code.

Some of the advantages of USE.IT are obvious. Specifications are formulated in AXES and therefore analyzable for certain desirable properties. The Analyzer ensures that AXES specifications are consistent, free of data and timing conflicts, and complete. It should be emphasized that this is done before implementation. Since the RAT automatically generates code, coding time is minimal.

USE.IT also provides the user with a library of AXES mechanisms. In this way a user defines systems drawing upon mechanisms found in the library. Of course, the user is not limited to those mechanisms, but may build his own mechanisms and store them in a library for use whenever needed.

The generality and portability of USE.IT allows it to produce "code" in languages other than the common or traditional ones. The RAT currently produces FORTRAN and PASCAL, but a POD RAT is also feasible. A POD RAT would take an analyzed AXES specification of a system, and automatically generate a POD description (System Description File) of that system. This System Description File would then be input to POD, which would then produce the system's performance evaluation.

The benefit of this approach is that the definition, i.e., AXES specification of a system, would have all the desirable properties (consistency, completeness, and correct data flow). USE.IT would then guarantee that the System Description File it automatically generates is logically complete. Moreover, a POD RAT together with, say, a FORTRAN RAT would enable a system to be tested using POD and then implemented in FORTRAN, all from one implementation-free HOS specification. Conversely, the POD tool could be used to help decide whether the system should be RATted into FORTRAN, or whether some other language would be more appropriate for its implementation.

3. POD - HOS INTERFACE

3.1 INTRODUCTION

HOS/AXES provides both a language for the definition of systems and a metalanguage for the definition of mechanisms that can themselves be used as a language for the definition of systems. In relation to a system like POD, the metalanguage aspect emerges as primary, because its own basic constructs can be defined formally as HOS mechanisms, thereby enhancing POD with all the benefits that that kind of formalization brings. Furthermore, a RAT that automatically generates POD descriptions from HOS specifications can be readily built once the semantics of POD is fully understood and made explicit.

3.2 HOS SPECIFICATION OF POD SEMANTICS

HOS characterizes all systems in terms of three fundamental units: data types, functions, and control structures; and the basic constructs of POD map naturally into this framework. The following examples demonstrate the manner in which POD semantics could be formalized with HOS.

Devices in POD, for example, comprise an HOS data type [5], a formal characterization of which is given in Figure 1. A user of POD must attribute to devices only those attributes which POD itself attributes to them, either explicitly or implicitly, and strict adherence to the specification in Figure 1 would guarantee that this was the case. The standard set of device attributes in POD is given in Figure 2, and each of these, as well as the device type, is reflected as a primitive operation in Figure 1. The default units and values in Figure 2 are omitted from Figure 1 solely in order to simplify the exposition, but they would be included in a more complete

```
DATA TYPE:  DEVICE;

PRIMITIVE OPERATIONS:

     device type = Device-Type (device);
     rational = Rate (device);
     list (of files) = Device-map (device);
     rational = Seeks (device);
     rational = Revolution-time (device);
     natural = Capacity (device);
     string (of characters) = Operation (natural,device);
     natural = Number-of-operations (device);
     formula = Time (natural, device);
     natural = Multiplicity (device);
     string (of characters) = Class (device);

AXIOMS:

WHERE REJECT IS A MEMBER OF EVERY TYPE;

     WHERE dev IS A DEVICE;
     WHERE nat IS A NATURAL;

     Not(Or(Equal(Device-type(dev),cpu),
            Equal(Device-Type(dev),disk),
            Equal(Device-Type(dev),server)))
                  = Equal(rate(dev),REJECT);

     Not(Or(Equal(Device-type(dev),defined),
            Equal(Device-type(dev),defined-cpu)))
                  = Equal(Operation(nat,dev),REJECT);

     Equal(Operation(nat,dev),REJECT)
                  = Equal (Time(nat,dev),REJECT);

     Not(Equal(Device-type(dev),memory))
                  = Equal(Capacity(dev),REJECT);

     Not(Equal(Device-type(dev),disk))
                  = Equal (Device-map (dev), REJECT);

     Not(Equal(Device-type(dev),disk))
                  = Equal(Seek(dev),REJECT);

     Not(Equal(Device-type(dev),disk))
                  = Equal(Revolution-time(dev),REJECT);

     Not(Or(Equal(Device-type(Dev),cpu),Equal(Device-type(dev),disk)))
                  = Equal(class(dev),REJECT);


     Not(Or(Equal(Device-type(dev),cpu),Equal(Device-type(dev),server)))
                  = Equal(Multiply(dev),REJECT);

     <(nat,Number-of-operation(dev))
                  = Equal(Operation(nat,dev),REJECT);

END DEVICE;
```


Figure 1: Data Type DEVICE in HOS

DEVICE TYPE	REQUIRED ATTRIBUTES	DEFAULT UNITS	OPTIONAL ATTRIBUTES	DEFAULT VALUES
CPU	RATE	MIPS	MULTIPLICITY CLASS	1
DISK	DEVICE_MAP RATE SEEK REVOLUTION_TIME	- CHAR/MSEC MSEC MSEC	CLASS	-
MEMORY	CAPACITY	WORDS		
SERVER	RATE	OPS/ MICROSECOND	MULTIPLICITY	1
DEFINED	OPERATION TIME	- MSEC		
DEFINED CPU	OPERATION TIME	- MSEC		

Figure 2: Device Attributes in POD (from [1])

specification of the data type. The number of operations a user-defined device has is not listed in Figure 2 as an attribute of POD devices, but it must be included in the HOS specification in order to formulate axioms that completely characterize the other primitive operations/attributes. HOS formalization thus brings to light a further attribute which POD and its users must implicitly take into account, even though POD does not explicitly recognize it.

The effect of the axioms in Figure 1 is to restrict each primitive operation, and thus each attribute, to exactly those device types that are appropriate, by specifying that its use rejects for devices of other device types. The first axiom states, for example, that the Rate primitive operation rejects for a device if and only if its device type is not cpu, disk, or server, indicating that only devices of these three types can properly be said to have rates in a POD description of a system, as figure 2 requires. The third axiom says that Operation rejects if and only if Time does and so, together with the second axiom, restricts both Operation and Time to be applicable only to devices of type defined or defined-cpu. The last axiom says that the n^{th} operation of a (user-defined) device exists if and only if n is less than or equal to the number of operations for that device. This is necessary in order to restrict the applicability of the earlier axioms that also contain Operation.

Most of the data types that provide inputs or outputs to the primitive operations in Figure 1 are already available in the general HOS library, but one of them, that of device types, is entirely specific to POD (in the present usage, at any rate). Since device types are, in fact, a kind of "object," whose members get associated with device "objects," they must be formally characterized in HOS as a data type, just as devices do themselves. A specification of the data type DEVICE TYPE is given in Figure 3.

```
DATA TYPE:  DEVICE TYPE;

PRIMITIVE OPERATIONS:

AXIOMS:

     WHERE cpu, disk, memory, server, defined,
       defined-cpu ARE CONSTANT DEVICE TYPES;
     WHERE dt IS A DEVICE TYPE;
     WHERE TRUE, FALSE ARE CONSTANT BOOLEANS;

     Or(Equal(dt,cpu),Equal(dt,disk),Equal(dt,memory),
        Equal(dt,server),Equal(dt,defined),
        Equal(dt,defined-cpu)) = True;

     Equal(cpu,disk) = False;
     Equal(cpu,memory) = False;
     Equal(cpu,server) = False;
     Equal(cpu,defined) = False;
     Equal(cpu,defined-cpu) = False;
     Equal(disk,memory) = False;
     Equal(disk,server) = False;
     Equal(disk,defined) = False;
     Equal(disk,defined-cpu) = False;
     Equal(memory,server) = False;
     Equal(memory,defined) = False;
     Equal(memory,defined-cpu) = False;
     Equal(server,defined) = False;
     Equal(server,defined-cpu) = False;
     Equal(defined,defined-cpu) = False;

     END DEVICE TYPE;
```

Figure 3: Data type DEVICE TYPE in HOS

Since device types are used solely to identify which attributes go with which kinds of devices, and since this has already been specified in Figure 1, data type DEVICE TYPE requires no primitive operations of its own, and so none are included in Figure 3. (Equal is a universal primitive operation, and Or is a boolean one, both available in the HOS library.) If reason were found for updating POD in some way that put device types to further use, then primitive operations and axioms that constrain them could be added to the data type specification to account for that. At present, however, the data type consists simply of six distinct members, identified in the first WHERE statement as the CONSTANT device types cpu, disk, memory, server, defined, and defined-cpu, the device types listed in Figure 2. These six device types are characterized in relation to each other in the axioms in Figure 3, the first of which says that any device type at all has to be one of the six, and the rest of which say that the six are, in fact, distinct. Any further properties that device types must be said to have can be introduced, as necessary, as further primitive operations with axioms to constrain them.

An HOS specification of data type FILE CATALOG is given in Figure 4 as a further example. The two essential components of a file catalog in POD are file names and record sizes, and these become primitive operations on the data type in HOS. The first axiom says that the length of each file name must be less than or equal to the value of some parameter to be specified by the user, perhaps in a formal characterization of a data type for file names. Length in Figure 4 is a primitive operation on strings available in the HOS library, but it could also be specified more abstractly as a primitive operation on a data type FILE NAME. The other two axioms specify that record sizes must fall within some range, saying that they must be greater than 0 and less than some user-supplied parameter value. These are sample axioms only. Further constraints are likely to be necesssary, especially in connection with file names.

```
DATA TYPE:  FILE CATALOG;

PRIMITIVE OPERATIONS:

   string(of characters) = File-name(natural,file catalog);
   natural = Record-size(natural,file catalog);

AXIOMS:

   WHERE fc IS A FILE CATALOG;
   WHERE nat IS A NATURAL;
   WHERE 0 IS A CONSTANT NATURAL;

   <(Length(File-name(nat,fc)),m) = True;
   <(0,Record-size(nat,fc)) = True;
   <(Record-size(nat,fc),n) = True;

END FILE CATALOG;
```

Figure 4: Data Type FILE CATALOG in HOS

Like devices and file catalogs, which comprise components of a configuration specification in POD, workloads are also characterizable most naturally in HOS as data types, as shown in Figure 5. Like devices, workloads have both attributes, shown in Figure 6, and types, which are themselves characterizable as a data type, as shown in Figure 7. Again, the attributes become primitive operations in HOS, with a further primitive operation that assigns each workload its type.

The components of POD module specifications, however, map into HOS not as data types, but as control maps, i.e., structured functions. The loop in Figure 8, for example, maps into the function tree in Figure 9, in which the data, subfunctions, and structural relations that are implicit in Figure 8 are indicated explicitly. Notice that the structure in Figure 9 Loop-Number-of-Images-Times, is not a primitive but a user defined control structure. A formal specification of this structure must be provided to fully explicate the intended behavior. In a similar manner other module specificatons (i.e. user templates) can be fully specified in terms of the true underlying semantics or meaning of these usage oriented templates. All of the POD modules must have one of these associated formal definitions associated with it if it is to be considered to be formally defined in the HOS sense of an AXES specification. Following is a walk through of the POD module as we understand it (see Figure 10).

The overall effect of the specification is to define a function, in the mathematical sense, called here Loop-Number-of-Images-Times. Choice of names is theoretically arbitrary in HOS, but good style involves making choices that enhance clarity and understanding. This function inputs values of a variable alpha and, perhaps, other input, such as the state of the relevant device. The way the function gets carried out is indicated by the three levels of decomposition into subfunctions. First, a counter, n, is initialized to 0 and then fed, along with alpha and input, into the function that comprises the main

```
DATA TYPE:  WORKLOAD;

PRIMITIVE OPERATIONS:

     workload type = Workload-type(workload);
     natural = Mpl(workload);
     list(of(job,percent)) = Job-stream(workload);
     natural = Arrival-rate(workload)
     natural = Think-time(workload);
     natural = Users(workload);
     Natural = Priority(workload);

AXIOMS:

     WHERE wl IS A WORKLOAD;
     WHERE 14 IS A CONSTANT NATURAL;

     Equal(Workload-type(wl),periodic)
                 = Equal(Mpl(wl),REJECT);
     Equal(Workload-type(wl),interactive)
                 = Equal(Job-stream(wl),REJECT);
     Equal(Workload-type(wl),interactive)
                 = Equal(Arrival-rate(we),REJECT);
     Equal(Workload-type(wl),cycle)
                 = Equal(Arrival-rate(wl),REJECT);
     Not(Equal(Workload-type(wl),interactive))
                 = Equal(Think-time(wl),RREJECT);
     Not(Equal(Workload-type(wl),interactive))
                 = Equal(Users(wl),REJECT);
     <(0,Priority(wl)) = True;
     <(Priority(wl),14) = True;

END WORKLOAD;
```

Figure 5: Data type WORKLOAD in HOS

WORKLOAD TYPE	REQUIRED ATTRIBUTES	DEFAULT UNITS	OPTIONAL ATTRIBUTES	DEFAULT VALUES
CYCLE	MPL JOB_STREAM	JOBS -	PRIORITY	LOWEST PRIORITY(0)
PERIODIC	JOB_STREAM and ARRIVAL_RATE or SOURCES	JOBS/HR -	DOMAIN_ID PRIORITY	- LOWEST PRIORITY(0)
TRANSACTION	JOB_STREAM and ARRIVAL_RATE or SOURCES MPL	- JOBS/HR - JOBS	PRIORITY DOMAIN_ID	LOWEST PRIORITY(0) -
INTERACTIVE	MPL THINK TIME USERS	JOBS SEC -	PRIORITY	LOWEST PRIORITY(0)

Figure 6: Workload Attributes in POD (from [1])

```
DATA TYPE:  WORKLOAD TYPE;

PRIMITIVE OPERATIONS:

AXIOMS:

    WHERE cycle, periodic, transaction,
    interactive ARE CONSTANT WORKLOAD TYPES;
    WHERE wt IS A WORKLOAD TYPE;

    Or(Equal(wt,cycle),Equal(wt,periodic),
       Equal(wt,transaction),Equal(wt,interactive)) = True;
    Equal(cycle,periodic) = False;
    Equal(cycle,transaction) = False;
    Equal(cycle,interactive) = False;
    Equal(periodic,transaction) = False;
    Equal(periodic,transaction) = False;
    Equal(transaction,interactive) = False;

END WORKLOAD TYPE;
```

Figure 7: Data type WORKLOAD TYPE in HOS

```
LOOP NUMBER OF IMAGES TIMES
     CALL TASK 2(ALPHA)
END LOOP
```

Figure 8: A POD LOOP Specification (from [1])

```
alpha', input' =  A_USE(alpha,input)
                  |
                  LOOP-NUMBER-OF-IMAGES-TIMES
alpha*, input* =  TASK2(alpha,input)
```

Figure 9: A Use of the Defined Structure LOOP-NUMBER-OF-IMAGES-TIMES formally defined in Figure (10)

subfunction, called here Loop-task2. K_0 is a universal primitive operation of HOS, a function that generates 0 as its output value no matter what its inputs are. Such a constant function K_i is available for use for any member i of any available data type. Second, a function called here Do-task updates alpha, the other input, and n and then determines whether to loop or stop, depending on the current value of n, now called n'. Changing a variable's value requires also changing the variable itself in HOS, even if only by adding a prime or asterisk, in order to maintain traceability of data and the possibility of static checking. Third, the counter gets updated and Task 2 actually gets carried out, after which, based on the counter's value, either Loop-task2 gets recalled for that value and the updated alpha and input or those values get retained as the final values. $Clone_1$ is another operation of HOS, one that produces one copy of its input, whatever that is, as its output, i.e., makes possible a further reference to its input.

The symbols CJ, J, I, and CO in Figure 9 indicate examples of HOS control structures [2,3], i.e., relations between functions and their subfunctions. CJ is the COJOIN structure, which indicates sequential execution and shared inputs. If n were the only input to Loop-task 2, then the CJ could be replaced with J, a purely sequential JOIN construct, in which the output of one subfunction is the only input to the other. Such a structure is involved, in fact, in the decomposition of Loop-task 2, whose subfunctions share no inputs, the output of one, namely, alpha*, input*, and n', being the only inputs to the other. I, which decomposes Do-task2 is a parallel INCLUDE structure, which partitions input and output lists and matches the sublists to each subfunction with no overlap. A COINCLUDE structure is also available, related to INCLUDE much as COJOIN is related to JOIN, but that structure is not needed for this example. CO is a COOR structure, indicating deterministic alternatives, which can also be used to

explicate the POD TEST construct, as shown in Figure 11 for the example in Figure 12. If $Clone_1$ (alpha*,input*) were replaced with $Identify^3_{1,2}$ (alpha*, input*, n'), then this COOR could be replaced with an OR structure, which requires each offspring to have exactly the same inputs as the parent function. $Identify^i_{j,\ldots k}$, for $j \leq k \leq i$, is another operation of HOS, the effect of which is to extract the $j^{th},\ldots$ and k^{th} members of a length-i input list. JOIN, OR, and INCLUDE comprise the three primitive control structures of HOS, out of which all other allowable structures can be defined. The lower-most occurrence of Loop-task2 re-invokes the named function for the indicated updated inputs, thereby creating the loop effect that is named, but not otherwise represented, in the POD notation. If this lower occurrence were replaced with some other function name not also occurring elsewhere in the tree, then this recursive effect would disappear.

It should be stressed that Figure 10 gives an explicit account of the semantics of the syntax in Figure 9. Figure 8 might seem easier to use, and indeed it is easier to use for one who is familiar with it, but the fact that so much of its meaning is only implicit makes it subject to misinterpretation, incorrect usage, side effects, and so on. The description in Figure 9, in contrast, is guaranteed to be logically correct, because it has a clearly identifiable meaning in terms of its formal definition in Figure 10 following the HOS rules, which eliminate interface errors and ensure correct modularization [2].

3.3 A RESOURCE ALLOCATION TOOL (RAT) TO GENERATE POD

If a system is described completely in HOS to begin with, POD code can be generated automatically from it with a RAT, as can code in any language for which a RAT is available [7]. We will use the

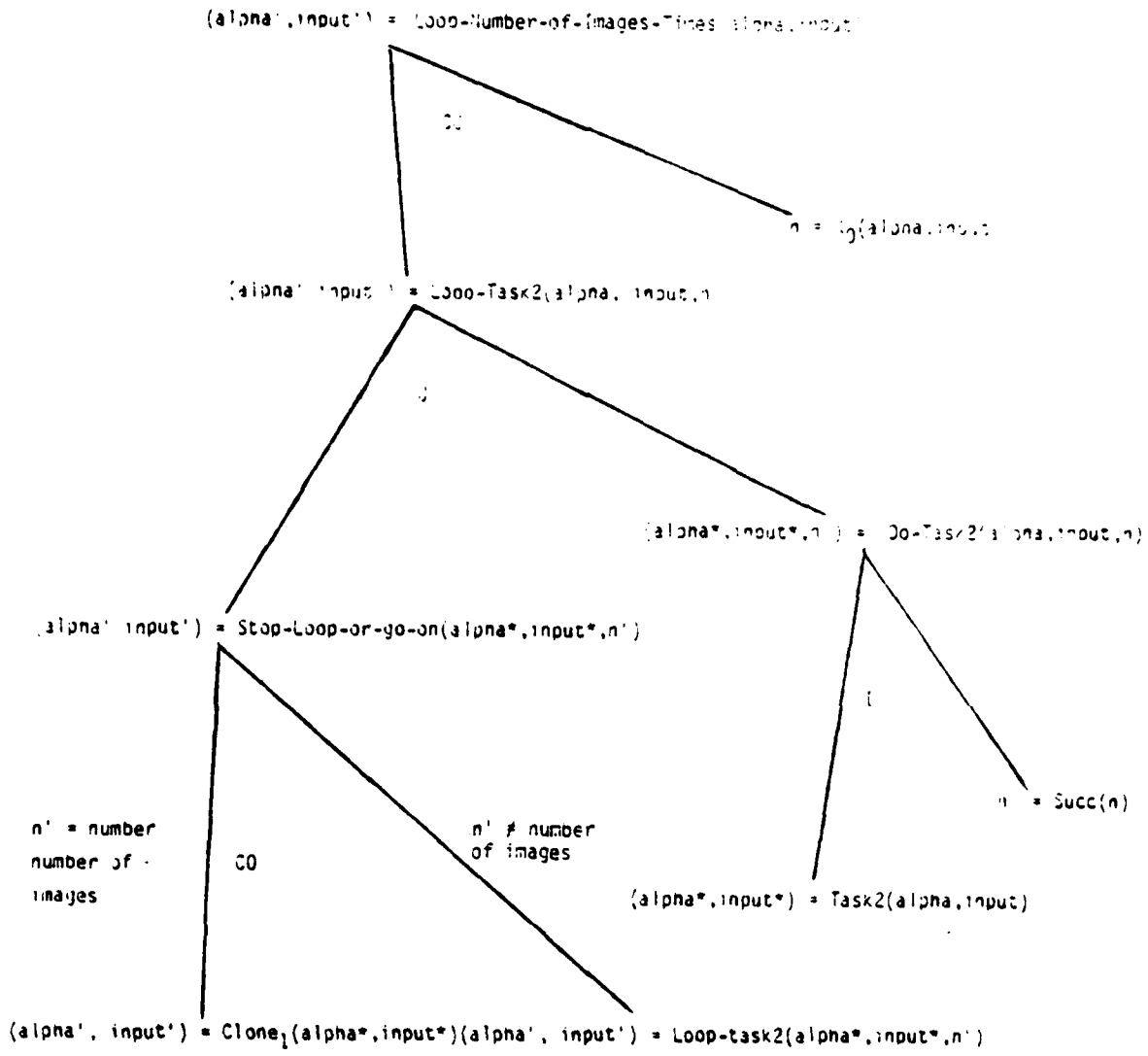

Figure 10: Formal Specification of Figure 9

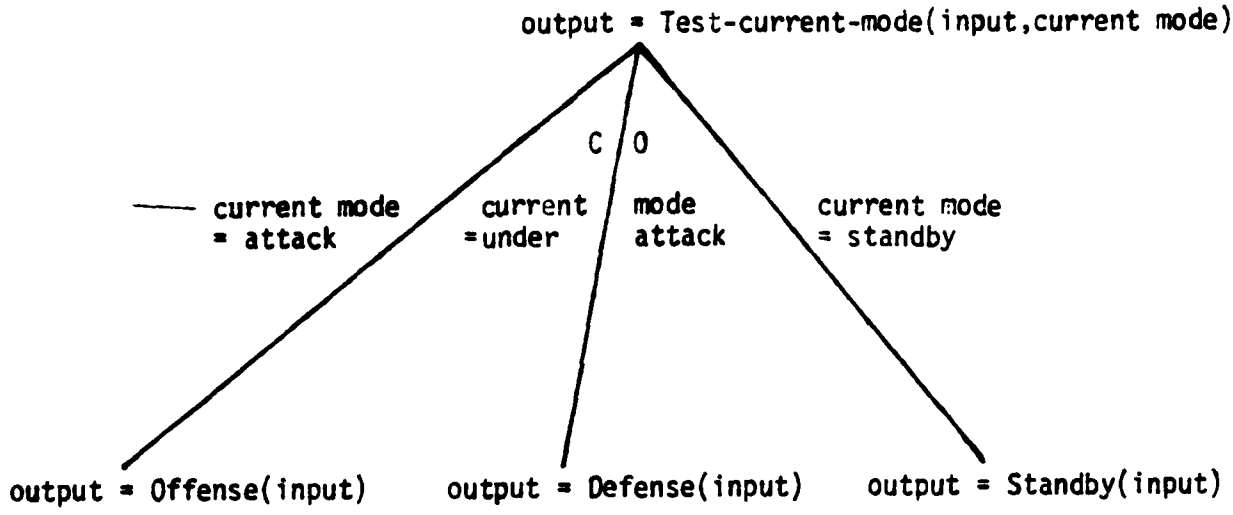

Figure 11: HOS Control Map for a POD CASE Statement

```
TEST CURRENT_MODE
 CASE 'STANDBY'
      CALL STANDBY

 CASE 'UNDER ATTACK'
      CALL DEFENSE

 CASE 'ATTACK'
      CALL OFFENSE
ENDTEST
```

Figure 12: The POD construct whose control map appears in in Figure 10 (from [1])

example in the appendix (which just so happens to be a specific POD specification) to show what it would mean to specify it in HOS functional notation and then show the connection between that specification and a POD RAT.

The example in the Appendix consists of specific values being given to various kinds of items: a file catalog, devices, modules, and so on; and each such item can be defined as a CONSTANT in HOS with a WHERE statement. An HOS specification of the file catalog in the example is given in Figure 13. For each natural number, the corresponding file name and record size are specified, exactly reflecting the information contained in the POD description, but making more explicit the fact that each of these depends both on the file catalog itself and on a choice of natural numbers.

The first device specification in the example can be translated into HOS as in Figure 14, which names the particular device and provides it with a type and with a value for the attribute that is required for it by Figures 1 and 2. The other devices are all disks and have the same values for their Rate and Seek attributes, so they can be specified either individually, as in Figure 15, or, more succinctly, by making use of an HOS version of the POD CLASS construct, as in Figure 16. A specific workload can be specified in terms of a defined structure in exactly the same way as shown in Figures 9 and 10, for the one in Figure 17.

Specific modules in POD can be expressed in HOS as control maps. The module Retrieve-record in the example, for instance, becomes the function tree in Figure 18, where the "..." is included only for perspicuity. As a further example, the LOOP module DISPLAY can be written in HOS as the control map in Figure 19, which has no TEST construct, as Figure 18 does, but has a recursive call to one its higher higher-level functions. Complete specifications of these

```
WHERE file catalog IS A CONSTANT FILE CATALOG;
WHERE F1,F2,F3,F4,L,A,D,SWD,SUCC ARE STRINGS OF CHARACTERS;
WHERE 1,2,3,4,5,6,7,8,9,100,50,36,200,450 ARE CONSTANT NATURALS;

    File-name(1,file catalog) = F1;
    File-name(2,file catalog) = F2;
    File-name(3,file catalog) = F3;
    File-name(4,file catalog) = F4;
    File-name(5,file catalog) = L;
    File-name(6,file catalog) = A;.
    File-name(7,file catalog) = D;
    File-name(8,file catalog) = SWD;
    File-name(9,file catalog) = SWC;
    Record-size(1,file catalog) = 100;
    Record-size(2,file catalog) = 100;
    Record-size(3,file catalog) = 100;
    Record-size(4,file catalog) = 100;
    Record-size(5,file catalog) = 50;
    Record-size(6,file catalog) = 36;
    Record-size(7,file catalog) = 200;
    Record-size(8,file catalog) = 450;
    Record-size(9,file catalog) = 450;

END file catalog;
```

Figure 13: HOS Specification of a Specific File Catalog

```
WHERE cpu IS A DEVICE TYPE;
WHERE 1.2 IS A CONSTANT RATIONAL;
WHERE central processor IS A CONSTANT DEVICE;
      Device-type (central processor) = cpu;
      Rate (central processor) = 1.2;


END central processor;
```

Figure 14: HOS Specification of a Specific Device

```
WHERE 20,30 ARE CONSTANT NATURALS;
WHERE disk IS A DEVICE TYPE;
WHERE disk1 IS A CONSTANT DEVICE;
     Device-type (disk1) = disk;
     Rate (disk1) = 100;
     Seek (disk1) = 20;
     Revolution-time (disk1) = 30
     Device-map (disk1) = (F1,SWD);
END disk1;

WHERE disk2 IS A CONSTANT DEVICE;
     Device-type (disk2) = disk;
     Rate (disk2) = 100;
     Seek (disk2) = 20;
     Revolution-time (disk2) = 30;
     Device-map (disk2) = (F2,SWC);
END disk2;

WHERE disk3 IS A CONSTANT DEVICE;
     Device-type (disk3) = disk;
     Rate (disk3) = 100;
     Seek (disk3) = 20;
     Revolution-time (disk3) = 30;
     Device-map (disk3) = (F3,L,A);
END disk3

WHERE disk4 IS A CONSTANT DEVICE;
     Device-type (disk4) = disk;
     Rate (disk4) = 100;
     Seek (disk4) = 20;
     Revolution-time(disk4) = 30;
     Device-map (disk4) = (F4,D);
END disk4;
```

Figure 15: HOS Specification of Some Specific Disks

```
WHERE disks IS A CONSTANT CLASS;
     Rate (disks) = 100;
     Seek (disks) = 20;
     Revolution-time (disks) = 30;

END disks;

WHERE disk1, disk2, disk3, disk4 ARE CONSTANT DEVICES;

     Device-type (disk1) = disk;
     Device-type (disk2) = disk;
     Device-type (disk3) = disk;
     Device-type (disk4) = disk;
     Class (disk1) = disks;
     Class (disk2) = disks;
     Class (disk3) = disks;
     Class (disk4) = disks;
     Device-map (disk1) = (F1,SWD);
     Device-map (disk2) = (F2,SWC);
     Device-map (disk3) = (F3 L,A);
     Device-map (disk4) = (F4,D);

END disk1, disk2, disk3, disk4;
```

Figure 16: HOS Specification of the CLASS of Disks in Figure 14

```
WHERE 5 and 10,000 ARE CONSTANT NATURALS;
WHERE 70% and 30% ARE CONSTANT PERCENTS;
WHERE data base usage IS A CONSTANT WORKLOAD;

     Workload-type (data base usage ) = transaction;
     Mpl (Data base usage) = 5;
     Arrival-rate (data base usage) = 10,000;
     Job-stream (data base usage) = ((Retrieve-record, 70%),
                                     (Update-record, 30%)),

END data base usage;
```

Figure 17: HOS Specification of a Specific Workload

modules might require more knowledge about the details of the data structures involved and perhaps other information as well that is only implicit in the POD example. In contrast to the POD module specification, an AXES defined structure definition makes explicit data, functions, control structures, and so on that are implicit in the POD specification's meaning, but not expressed in the notation. An AXES specification has two components in a user defined structure. The first is a simple use oriented syntax (as in Figure 9) stating the missing variable information needs of the full definition (as in Figure 10). The first is comparable to the POD module specification. The second is a full system definition which is hidden to simplify the actual usage of that definition. This second component, the definition of the meaning, must be available for a user to really understand what these POD module specifications really do.

In general, performance information can be derived from a control map in conjunction with information about the hardware and resident software (and so on) on which it is to be implemented, [6] in accordance with the general operation template

performance information = POD-RAT(control map, hardware, resident software,...).
(expressed in POD notation) (all expressed in HOS/AXES)

For the control map in Figure 18, for example, the INCLUDE structure in the lower right indicates the <u>possibility</u> of a parallel implementation. If the hardware and resident software permit this possibility to be realized, then the structure can be implemented in that way; otherwise, the two functions Access-record and Succ can be implemented in sequence, and in either order, because the possibility of parallelism implies non-dependence. In either case, performance time, say, of these two primitive operations can be determined from the hardware/software input and that of Do-access-record determined from that and the choice of implementation of the structure. Similarly, the

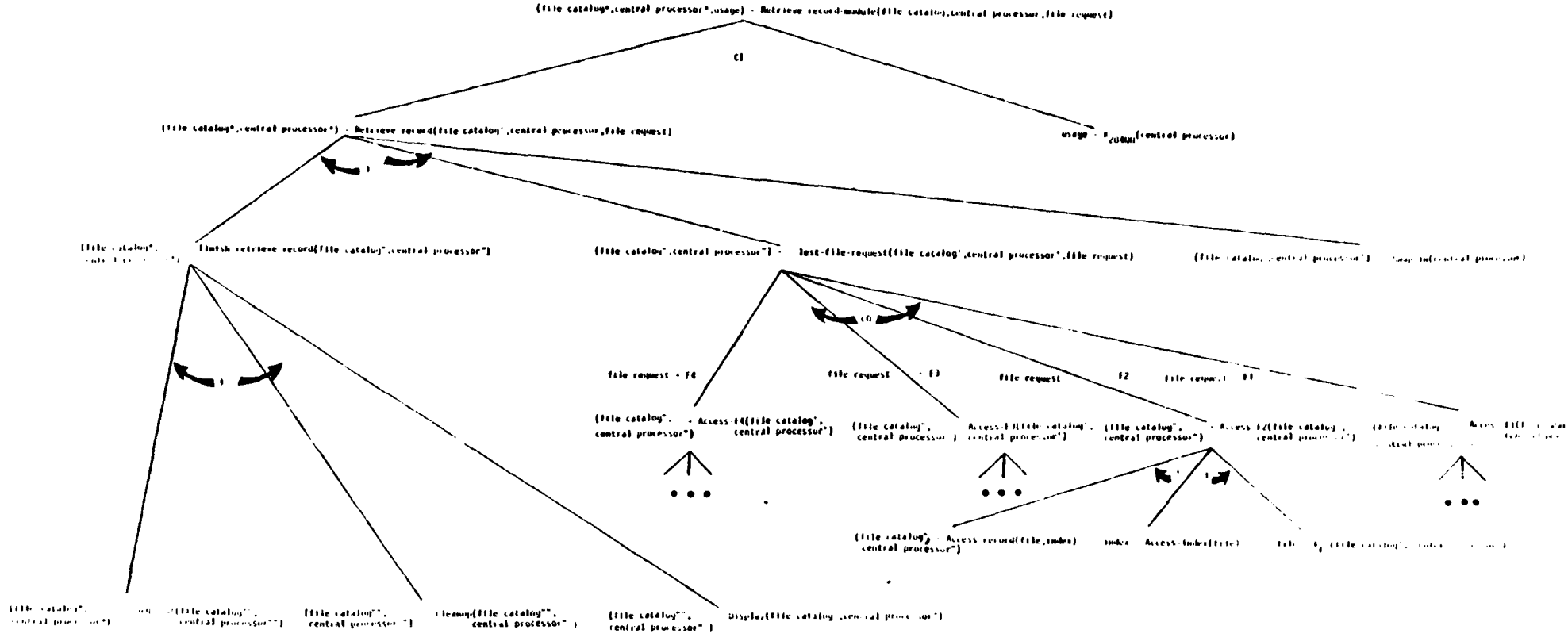

Figure 18: IIOS Specification of a Specific Module

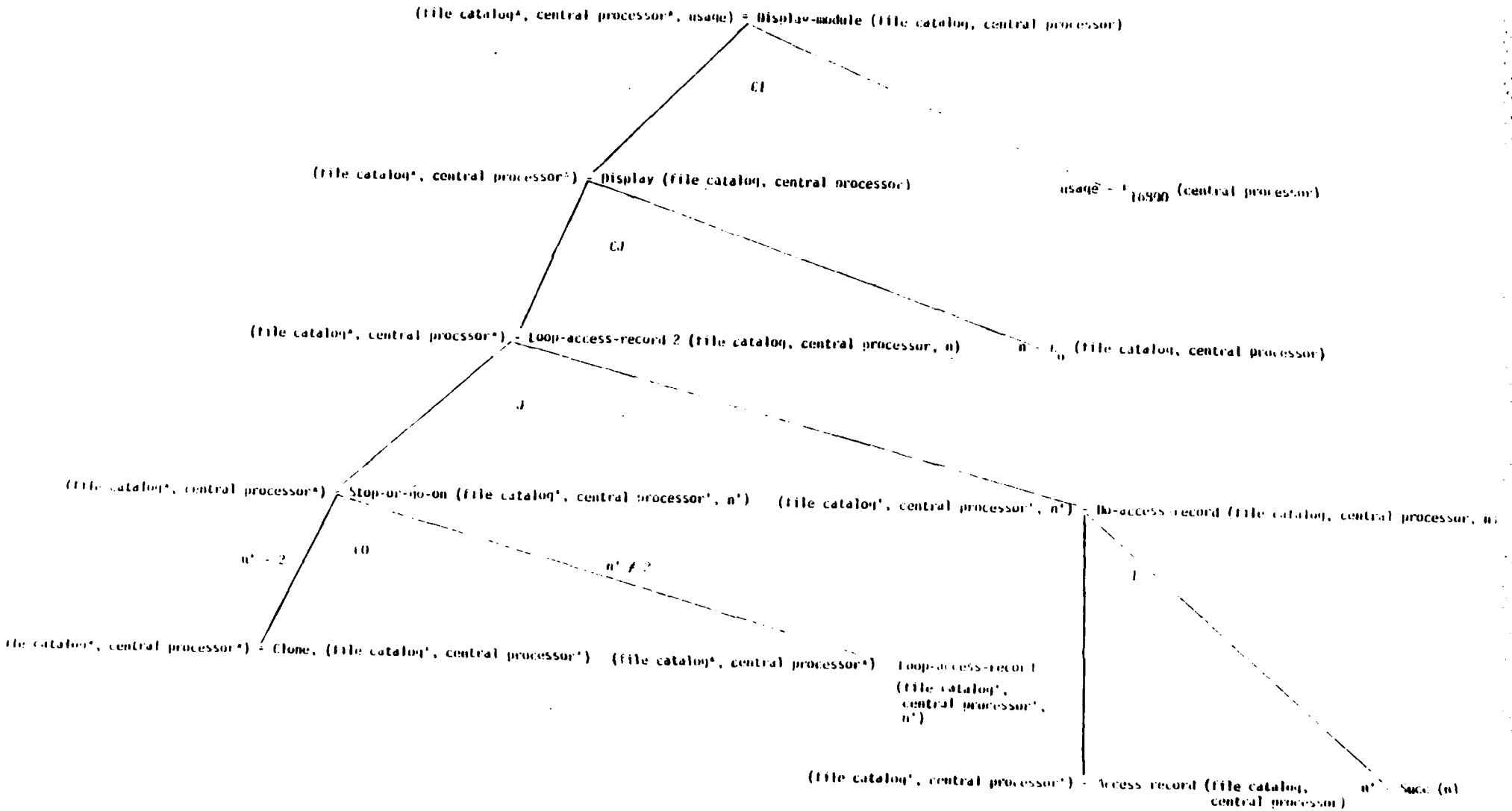

FIGURE 19: HOS SPECIFICATION OF A SPECIFIC RECURSIVE (LOOP) MODULE

COOR structure on the lower left tells us that only one of the sub-functions is ever run on a particular performance pass and thus that only one processor need ever be made available for executing the structure as a whole, i.e., the compound operation Stop_or_go_on. Once everything involved is expressed clearly in HOS/AXES--i.e., system specification, hardware, resident software, and so on--, all of the relevant information concerning performance time, efficiency, etc., can be readily determined by a RAT and expressed in POD formalism for processing as usual by the POD tool.

Rather than writing systems directly in POD, one can write them instead in HOS and then generate the POD code automatically. The advantage of this way of doing things is that the same HOS specifications can also be input to other RATS, such as those for FORTRAN, PASCAL, or other programming languages. Once a single HOS specification is completed, any number of other versions of it can be automatically generated, for any language for which a RAT has been built. Furthermore, the POD tool can be used to evaluate which RAT should be used to implement a particular system, since RATs themselves can be taken as inputs to the POD-RAT function. A system that involves a lot of concurrency, for example, should be RATted into an implementation language that best supports that facility, whereas one that has none should be RATted into a language that does not, in order to optimize both resource usage and performance characteristics. The use of a POD RAT in making these decisions would optimize those decisions, thereby enhancing system performance [6,7].

3.4 RECOMMENDATIONS FOR FUTURE WORK

The single most important task that should be undertaken as a further development of the HOS-POD interface is the construction of a POD RAT. Such a RAT would make possible the automatic evaluation of systems expressed in HOS/AXES in order to determine their optimal

implementation environment by generating POD descriptions to be input to the POD tool. Writing systems in HOS would make further coding unnecessary, since a single specification could be used both for evaluating and for implementing systems, as well as for choosing the optimal implementation.

A second task would be developing a complete HOS specification of the full POD semantics, both in order to make explicit the full power of POD and also to enhance the building of a POD RAT. A POD RAT could be built without such a full specification, but its existence would simplify and enhance the building of a RAT. Ideally, this task would be included as a major subtask in the beginning stages of a RAT development.

REFERENCES

[1] J. P. Buzen, G. B. Giacone, D. E. Hall, P. S. Mager, R. T. Williams, "Performance Oriented Design POD Reference Manual," BGS Systems, Inc., P.O. Box 128, Lincoln, MA, September, 1981.

[2] M. Hamilton and S. Zeldin, "Higher Order Software, Inc. - A Methodology for Defining Software," IEEE Transactions on Software Engineering SE-2 (1), 9-32, 1976.

[3] M. Hamilton and S. Zeldin, "The Relationship Between Design and Verification," The Journal of Systems and Software, Elsevier North Holland, Inc. New York, New York, Volume 1, No. 1, 1979.

[4] Higher Order Software, Inc., Cambridge, Massachusetts, "USE.IT Reference Manual," Reference Manual No. 6, May 1982.

[5] S. Cushing, "Algebraic Specification of Data Types in Higher Order Software (HOS)," Proceedings Eleventh Hawaii International Conference on System Sciences, Volume 1, University of Hawaii, Honolulu, Hawaii, January 5 - 6, 1978.

[6] M. Hamilton and S. Zeldin, "Properties of User Requirements", Formal Models and Practical Tools for Information Systems Design; edited by Hans-Jochjen Schneider, North Holland Publishing Co., April 1979.

[7] M. Hamilton and S. Zeldin, "A Functional Approach To The Life Cycle Model: Towards a Development Support System for DOT," (Part 1 of 2 Parts), prepared for SDC Integrated Services, Inc. McLean, Virginia, August 1981.

APPENDIX A

DEMONSTRATING POD FUNCTIONALITY

(FROM [1])

APPENDIX A: DEMONSTRATING POD FUNCTIONALITY

Background

This example involves the analysis of an idealized on-line system for information retrieval and updating. Systems of this type are used within the Navy for such purposes as maintaining information on the location of ships, the status of personnel, the availability of logistic support facilities, and so on.

We assume in this particular example that the information within the system is organized into four files: F1, F2, F3, F4. An operator at a terminal can, in a single transaction, either retrieve or change (update) a record from one of these files. Thus, the system supports eight separate transaction types:

R1 Retrieve a record from file F1
R2 Retrieve a record from file F2
R3 Retrieve a record from file F3
R4 Retrieve a record from file F4
C1 Change (update) a record in file F1
C2 Change (update) a record in file F2
C3 Change (update) a record in file F3
C4 Change (update) a record in file F4

In transaction types R1 - R4, the operator types in a record identifier (a key), and the system retrieves the appropriate record and displays its contents on a screen. Thus, files F1 - F4 function as indexed files.

In order to process a transaction, it is necessary to swap the appropriate processing routines into main memory, access one of the four data files (F1 - F4), and also access a log file (L), an accounting file (A) and a display file (D). The average number of accesses per transaction to each of the files is given below:

	F1	F2	F3	F4	L	A	D	SWD	SWC
R1	2	0	0	0	1	1	2	2	1
R2	0	2	0	0	1	1	2	2	1
R3	0	0	2	0	1	1	2	2	1
R4	0	0	0	2	1	1	2	2	1
C1	2	0	0	0	1	1	2	2	1
C2	0	2	0	0	1	1	2	2	1
C3	0	0	2	0	1	1	2	2	1
C4	0	0	0	2	1	1	2	2	1

File accesses/transaction

Note that there are two swap files: the data swap file, SWD, and the code swap file, SWC. Upon initiation of a transaction both data (from SWD) and code (from SWC) are swapped in. Upon termination of a transaction only the data is swapped out (to SWD) since the code is pure. Thus each transaction accesses SWD twice and SWC once.

In the case of the "retrieve" transactions (R1 - R4), the two accesses to the data files (F1 - F4) represent one access to the index and one access to the data itself. In the case of "change" transactions (C1 - C4), the system is designed so that a change must always follow a retrieve and will always alter the record that has just been retrieved. Thus, the access to the index is eliminated. However, it is necessary in this case to carry out a second read access after the record has been written to verify the fact that there were no errors in the write operation. The value that is actually read from the disk after writing is displayed on the screen as part of the "change" transaction.

Performance Data

Assume that the system being evaluated has four high speed disk drives, each with a transfer rate of 100 bytes/msec, an average seek time of 20 msec and a revolution time of 30 msec. Also suppose that the files are mapped onto the disks as follows:

DISK 1: F1, SWD
DISK 2: F2, SWC
DISK 3: F3, L, A
DISK 4: F4, D

Assume that the following information is also known:

Transaction arrival rate = 10000 per hour
Retrieval percentage = 70%
(R1 - R4 all occur in equal proportions)
Change percentage = 30%
(C1 - C4 all occur in equal proportions)
Number of message regions = 5

Initial POD Specification

The facilities of POD will be demonstrated by addressing the following questions:

1. Someone has suggested that it might be possible to improve average data base transaction response time by making the index portions of files F1 - F4 memory resident. This would save one disk access in transaction types R1 - R4. In addition, by eliminating the CPU overhead necessary to initiate the index access (and carrying out other optimization steps while modifying the code), assume it is possible to reduce the average CPU time required for the module accessing the index by 90%. Assume also that it is necessary to reduce number of message regions by two in order to make the index portions of the files memory resident. How will this suggestion affect overall response time?

2. Assume that the load on the system is expected to grow to 14350 transactions per hour in the next year. What will the response time be at that time for both the memory resident and the disk resident options?

3. Assume that a faster CPU will be available when the 14350 transaction per hour load is reached. The faster CPU will execute instructions in an average of 10% faster than the current CPU. What will the response times be for each option if the faster CPU is used?

The following is a sample POD System Description File that contains sufficient detail to address these issues. It should be noted that liberties have been taken in selecting the CPU processing requirements of the modules and the file organization characteristics. In general, such data would have to be collected and supplied by the user.

```
GLOBAL.DECLARATION
   PARAMETER
      FILE_REQUEST = 'F1' 25%
                     'F2' 25%
                     'F3' 25%
                     'F4' 25%
   END
END


CONFIGURATION.SPECIFICATION
   FILE_CATALOG
      NAME=F1, RECORD_SIZE=100
      NAME=F2, RECORD_SIZE=100
      NAME=F3, RECORD_SIZE=100
      NAME=F4, RECORD_SIZE=100
      NAME=L, RECORD_SIZE=50
      NAME=A, RECORD_SIZE=36
      NAME=D, RECORD_SIZE=200
      NAME=SWD, RECORD_SIZE=450
      NAME=SWC, RECORD_SIZE=450
   END

   DEVICE CENTRAL_PROCESSOR TYPE = CPU
      RATE = 1.2 &MIP
   END

   DEVICE DISK1 TYPE = DISK
      RATE = 100 &KBYTES/SEC
      SEEK = 20 &MSEC
      REVOLUTION_TIME = 30 &MSEC
      DEVICE_MAP = F1, SWD
   END

   DEVICE DISK2 TYPE = DISK
      RATE = 100 &KBYTES/SEC
      SEEK = 20 &MSEC
      REVOLUTION_TIME = 30 &MSEC
      DEVICE_MAP = F2, SWC
   END

   DEVICE DISK3 TYPE = DISK
      RATE = 100 &KBYTES/SEC
      SEEK = 20 &MSEC
      REVOLUTION_TIME = 30 &MSEC
      DEVICE_MAP = F3, L, A
   END

   DEVICE DISK4 TYPE = DISK
      RATE = 100 &KBYTES/SEC
      SEEK = 20 &MSEC
      REVOLUTION_TIME = 30 &MSEC
```

```
      DEVICE_MAP = F4, D
   END
END

WORKLOAD.SPECIFICATION
   WORKLOAD DATA_BASE_USAGE TYPE = TRANSACTION
      MPL = 5
      ARRIVAL_RATE = 10000 X/HR
      JOB_STREAM = RETRIEVE_RECORD 70%
                   UPDATE_RECORD 30%
   END
END

MODULE.SPECIFICATION
&RETRIEVE A DATA BASE RECORD:
   MODULE RETRIEVE_RECORD
      EST  CENTRAL_PROCESSOR USAGE = 50400 INS
      CALL SWAP_IN
      TEST FILE_REQUEST
         CASE 'F1'
            CALL ACCESS_INDEX('F1')
            CALL ACCESS_RECORD('F1')
         CASE 'F2'
            CALL ACCESS_INDEX('F2')
            CALL ACCESS_RECORD('F2')
         CASE 'F3'
            CALL ACCESS_INDEX('F3')
            CALL ACCESS_RECORD('F3')
         CASE 'F4'
            CALL ACCESS_INDEX('F4')
            CALL ACCESS_RECORD('F4')
      ENDTEST
      CALL DISPLAY
      CALL CLEANUP
      CALL SWAP_OUT
   END


&UPDATE A DATA BASE RECORD:
   MODULE UPDATE_RECORD
      EST  CENTRAL_PROCESSOR USAGE = 50400 INS
      CALL SWAP_IN
      TEST FILE_REQUEST
         CASE 'F1'
            CALL WRITE_RECORD('F1')
            CALL VERIFY_WRITE('F1')
         CASE 'F2'
            CALL WRITE_RECORD('F2')
            CALL VERIFY_WRITE('F2')
         CASE 'F3'
            CALL WRITE_RECORD('F3')
            CALL VERIFY_WRITE('F3')
         CASE 'F4'
            CALL WRITE_RECORD('F4')
```

```
            CALL VERIFY_WRITE('F4')
         ENDTEST
         CALL DISPLAY
         CALL CLEANUP
         CALL SWAP_OUT
      END

   &ACCESS A RECORD FROM THE SPECIFIED FILE:
      MODULE ACCESS_RECORD(FILE)
         EST  CENTRAL_PROCESSOR USAGE = 19200 INS
         TEST FILE
            CASE 'F1'
               EST  F1 USAGE = 1 READ
            CASE 'F2'
               EST  F2 USAGE = 1 READ
            CASE 'F3'
               EST  F3 USAGE = 1 READ
            CASE 'F4'
               EST  F4 USAGE = 1 READ
            CASE 'D'
               EST  D USAGE = 1 READ
         ENDTEST
      END

   &ACCESS THE INDEX ON THE SPECIFIED FILE:
      MODULE ACCESS_INDEX(FILE)
         EST CENTRAL_PROCESSOR USAGE = 90480 INS
         TEST FILE
            CASE 'F1'
               EST  F1 USAGE = 1 READ
            CASE 'F2'
               EST  F2 USAGE = 1 READ
            CASE 'F3'
               EST  F3 USAGE = 1 READ
            CASE 'F4'
               EST  F4 USAGE = 1 READ
         ENDTEST
   END

   &DISPLAY THE RETRIEVED RECORD:
      MODULE DISPLAY
         EST  CENTRAL_PROCESSOR USAGE = 16800 INS
         LOOP 2 TIMES
            CALL ACCESS_RECORD('D')
         ENDLOOP
      END



   &INFORM THE LOG AND ACCOUNTING FILES OF THE TRANSACTION ACTIVITY:
      MODULE CLEANUP
         EST  CENTRAL_PROCESSOR USAGE = 3600 INS
         CALL WRITE_RECORD('A')
         CALL WRITE_RECORD('L')
```

```
   END

&WRITE A RECORD TO THE SPECIFIED FILE
   MODULE WRITE_RECORD(FILE
      EST  CENTRAL_PROCESSOR USAGE = 21700 INS
      TEST FILE
         CASE 'F1'
            EST  F1 USAGE = 1 WRITE
         CASE 'F2'
            EST  F2 USAGE = 1 WRITE
         CASE 'F3'
            EST  F3 USAGE = 1 WRITE
         CASE 'F4'
            EST  F4 USAGE = 1 WRITE
         CASE 'A'
            EST  A USAGE = 1 WRITE
         CASE 'L'
            EST  L USAGE = 1 WRITE
      ENDTEST
   END

&VERIFY THAT THE CHANGE TO THE FILE IS CORRECT:
   MODULE VERIFY_WRITE(FILE)
      EST  CENTRAL_PROCESSOR USAGE = 29800 INS
      TEST FILE
         CASE 'F1'
            EST  F1 USAGE = 1 READ
         CASE 'F2'
            EST  F2 USAGE = 1 READ
         CASE 'F3'
            EST  F3 USAGE = 1 READ
         CASE 'F4'
            EST  F4 USAGE = 1 READ
      ENDTEST
   END


&SWAP IN THE REQUIRED DATA AND CODE
   MODULE SWAP_IN
      EST  CENTRAL_PROCESSOR USAGE = 27900 INS
      EST  SWD USAGE = 1 READ
      EST  SWC USAGE = 1 READ
   END

&SWAP OUT THE DATA
   MODULE SWAP_OUT
      EST  CENTRAL_PROCESSOR USAGE = 27900 INS
      EST  SWD USAGE  = 1 WRITE
   END

END
```

The POD System Description File represents the basic specification of the system with the disk resident indices. Specifically, the Global Declaration Section defines FILE_REQUEST to account for the variability in an arriving transaction's request to one of the files F1 - F4. The Configuration Specification Section describes the organization of all of the files, the mapping of the files to the disks, and the characteristics of the hardware. The Workload Specification Section defines a single workload, DATA_BASE_USAGE, which has a load of 10,000 requests/hour of which 70% invoke module RETRIEVE_RECORD and 30% invoke module UPDATE_RECORD. This represents transactions R1-R4 and C1-C4 respectively. The MPL parameter constrains the number of transactions the system processes in parallel. The Module Specification Section represents the system software. The function of each module is described through the comments in the System Description File.

Note, the above System Description File portrays the system with the disk resident index option only. In order to portray the system with the memory resident indices, two changes to the System Description File are required. First the workload specifications must indicate the reduced number of message regions. Second, the ACCESS_INDEX module need not reference files F1-F4 for the indices and incurs a savings of 90% in its CPU processing requirement. These changes are depicted below.

Modified Workload Specification Section:

```
WORKLOAD.SPECIFICATION
   WORKLOAD DATA_BASE_USAGE TYPE = TRANSACTION
      MPL = 3
      ARRIVAL_RATE = 10000 &/HR
      JOB_STREAM = RETRIEVE_RECORD 70%
                   UPDATE_RECORD 30%
   END
END
```

Modified ACCESS_INDEX Module:

```
MODULE ACCESS_INDEX
   EST CENTRAL_PROCESSOR USAGE = 2048 INS
END
```

Also, CALLs to the module ACCESS_INDEX were changed to remove the passed parameter since the parameter is no longer required.

Note, in order to distinguish between these two System Description Files in the remainder of this discussion, we refer to the file representing the disk resident option and the memory resident option by file numbers 46 and 47 respectively. We are now in a position to invoke the interactive facilities of POD to investigate the issues cited above. The interactive session first investigates the performance of the system with the disk resident index option and next with the memory resident index option.

```
READ 46
 INPUT FILE COMPLETED

GO


*** PRINCIPAL RESULTS ***

WORKLOAD                    RESPONSE TIME   RESIDENCY TIME    THROUGHPUT     % CPU

  1 DATA_BASE_USAGE            1.16 SEC        0.97 SEC   10000. PER HOUR   64.6%
                                                 TOTAL CPU UTILIZATION =   64.6%


SET DATA_BASE_USAGE ARRIVAL_RATE = 14350

GO


*** PRINCIPAL RESULTS ***

WORKLOAD                    RESPONSE TIME   RESIDENCY TIME    THROUGHPUT     % CPU

  1 DATA_BASE_USAGE           26.04 SEC        1.23 SEC   14350. PER HOUR   92.7%
                                                 TOTAL CPU UTILIZATION =   92.7%


SET CENTRAL_PROCESSOR RATE = OLD * 1.1

GO


*** PRINCIPAL RESULTS ***

WORKLOAD                    RESPONSE TIME   RESIDENCY TIME    THROUGHPUT     % CPU

  1 DATA_BASE_USAGE           [illegible]     [illegible]   [illegible] PER HOUR   84.3%
                                                 TOTAL CPU UTILIZATION =   84.3%
```

```
READ 47
 INPUT FILE COMPLETED
GO

*** PRINCIPAL RESULTS ***

WORKLOAD                    RESPONSE TIME   RESIDENCY TIME    THROUGHPUT     % CPU

  1 DATA_BASE_USAGE             1.01 SEC        0.67 SEC   10000. PER HOUR   51.4%
                                                TOTAL CPU UTILIZATION =      51.4%

SET DATA_BASE_USAGE ARRIVAL_RATE = 14750
GO

*** PRINCIPAL RESULTS ***

WORKLOAD                    RESPONSE TIME   RESIDENCY TIME    THROUGHPUT     % CPU

  1 DATA_BASE_USAGE            20.09 SEC        0.74 SEC   14750. PER HOUR   75.8%
                                                TOTAL CPU UTILIZATION =      75.8%

SET CENTRAL_PROCESSOR RATE = OLD * 1.1
GO

*** PRINCIPAL RESULTS ***

WORKLOAD                    RESPONSE TIME   RESIDENCY TIME    THROUGHPUT     % CPU

  1 DATA_BASE_USAGE             4.14 SEC        0.70 SEC   14750. PER HOUR   67.1%
                                                TOTAL CPU UTILIZATION =      67.1%
```

The data produced from this POD interactive session is summarized in Figure A-1.

ENVIRONMENT	LOAD (per hr)	AVERAGE RESPONSE TIME (per sec)	CPU UTILIZATION
DISK RESIDENT INDEXES	10,000	1.16	64.6
MEMORY RESIDENT INDEXES	10,000	1.01	51.4
DISK RESIDENT INDEXES	14,350	26.04	92.7
MEMORY RESIDENT INDEXES	14,350	20.09	73.8
DISK RESIDENT INDEXES PLUS CPU UPGRADE	14,350	3.26	84.3
MEMORY RESIDENT INDEXES PLUS CPU UPGRADE	14,350	4.14	57.1

Figure A-1: Summary of Results

The results show:

- The system using disk resident indexes, while producing acceptable levels of response currently, will be incapable of processing the expected future load with acceptable response delay.
- Slight reduction in response time is achieved by using memory resident indexes over the disk resident indexes in the current time frame. With the heavier workload anticipated in the future time frame, the reverse can be expected.
- In the future time frame response time will be unacceptable unless the faster CPU is obtained, regardless of the index option selected.