```
PPPP   H   H   III   L        000    SSSS   000   PPPP   H   H  Y   Y
P   P  H   H    I    L       0   0  S      0   0  P   P  H   H  Y   Y
P   P  H   H    I    L       0   0  S      0   0  P   P  H   H   Y Y
PPPP   HHHHH    I    L       0   0   SSS   0   0  PPPP   HHHHH    Y
P      H   H    I    L       0   0      S  0   0  P      H   H    Y
P      H   H    I    L       0   0      S  0   0  P      H   H    Y
P      H   H   III   LLLLL    000   SSSS    000   P      H   H    Y
```

```
        333
       3   3
           3
          3
         3
        3
       3   3
  ..    333
  ..
```

```
*START* Job PHILOS Req #155 for EGB     Date 28-Sep-82 22:04:53 Monitor: Rational
File RM:<RPE.DOC>PHILOSOPHY..3, created: 24-Aug-82 14:06:40
        printed: 28-Sep-82 22:04:53
Job parameters: Request created:28-Sep-82 22:03:03   Page limit:36   Forms:NORMAL
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:A
```

Introduction

This document discusses the basic concepts and philosophy of the R1000 operating system and programming environment.

Traditional architectures and programming environments are incompatible with modern programming methodologies and Ada. Conventional Operating Systems deal with the wrong problems on the wrong level of abstraction; they are inadequate to support our environment design goals. It is doubtful that someone can build an effective programming environment on top of a standard OS.

Any programming environment will (hopefully) do more than a standard OS, this will add quite a bit of overhead to the host system. The fact that the R-1000 is designed to support Ada and programming environment function will offer a tremendous performance advantage to our customers.

A) Goals and Objectives

The R1000 design was guided by the recognition of the importance of program reliability and maintenance; central goals were:

-- mitigate the software crisis
-- reduce software life cycle costs
-- improve software/system reliability

The keys to successfully achieving these goal are

-- concern for programming as a human activity
-- user friendly
-- improve productivity
-- consistent use of Ada on all levels of interaction
-- Breadth of scope and applicability
-- tight integration of capabilities
-- efficiency
-- efficient real time execution of Ada
-- portability to upgraded systems

Keys to achieving these objectives are:
- to fully utilize facilities provided by the advanced R1000 architecture
- to use Ada concepts and semantics where ever possible.

Proper use of the R1000 architecture allows great simplicity and flexibility of the R1000 software. For example, the distinction of data stored in memory and data stored on disk files disappears on the R1000. This uniformity has numerous significant advantages, e.g. data can be accessed randomly, no IO programming is required, arbitrary pointer structures can be stored easily, and protection is provided by normal Ada scope rules.

The use of Ada concepts throughout the design resulted in an environment that is understandable and can be explained in terms of Ada semantics. Only very few additional concepts are required.

The R1000 is not the sum of programming environment, Compiler, Architecture, Hardware. Rather, the R1000 is an integrated system. The synergism possible through the integration of Ada and the R-1000 system results in a system not conceivable on conventional architectures.

The programming environment design must take into account the R-1000 architecture, our reliance on Ada as the definer of our market niche, and the needs/desires of the potential programming environment user community. The design must be sufficiently ahead of the "state of the art" to provide product differentiation yet it must be, in its minimal form, implementable by the late fall of 1982. We must look, explicitly, at the designs offered or likely to be offered by competitors offering systems that will be alternatives to or compete with the Company's system. The extensibility of the environment both for adding tools developed at RMI and our ability to integrate tools developed elsewhere is an important consideration. For efficient use of the programming environment, in particular by novice users it is necessary to not simply adding tools but to add knowledge about tools as well. The programming environment should be capable of helping the user access its component tools in an effective and productive manner.

The programming environment will ease the transition from one phase to another of the design, development, and maintenance of software.

The programming environment design must be responsive to the needs of all individuals involved in the software design and development over the life cycle. We must be sensitive to the needs of managers and provide for tools that will enable managers to benefit from out programming environment.

B) User view of the System

We recognize the fact that the programming environment may be advanced to a point where the user may not understand or feel familiar with its operation. The gap in knowledge responsible for this attitude is bridged from two directions. Our own user interface and design must be sensitive to the user's perspective. Second, we must educate the user, through documentation, articles in the popular journals, endorsements by industry leaders and respected computer scientists, articles published documenting initial customers favorable experiences with the system, to appreciate the system.

One important point is that the complete system is written in Ada. The user view of the system is Ada -- Ada semantics explains the operation of all features and tools of the system.

Since everything is written in Ada users will be able to easily customize the whole system. The question seems to be to what extend we should allow modification of the RMI code. What are legal changes; what changes void our software maintenance; what part of the source of the programming environment is released to the customer. These are important questions, recently articles have appeared demanding companies warrant software. The fact that the programming environment will be capable of leading the user to learn more about Ada through use of its facilities is a strong selling point--to management as well as the software engineer. No time is lost learning the system, everything (almost) learned for the sake of the system is consistent with things the person would have to learn to become more familiar with Ada. We can rely on people within the Company for some user feedback but we must also attempt, at the appropriate time, to expose select members of the "industry " to the programming environment and ensure the market acceptance of the environment. One idea is to implement some type of ETEACH routine to serve as system documentation (?) at the executive level, and to allow the user to get some quick REWARDING experience the first time logging on.

The following system structure will guarantee the above goals:

A R1000 machine runs only one program, the package R1000. This package contains the package SYSTEM and packages for individual users. The SYSTEM in turn provides other program units, such as packages of utility routines, accounting tasks, device controllers, and components of the programming environment. Each user of the system is represented by a package. A user may himself create users within his own package, corresponding to subaccounts on other systems. The overall system view is illustrated in figure 1.

All objects listed in the visible part of a user package are visible to other users declared in the enclosing scope; user packages and the system conform to Ada semantics in all respects. Ada scope rules provide a powerful means for protection.

All program objects created by a user are subunits of his user package. In particular, there is no need to store programs as data objects -- programs are part of the overall system. A programmer may store data in local variables of his package (or sub units), thus, the notion of file in a conventional system is replaced by variables in Ada.

Of course, variables provide only a primitive way of storing data. Ada allows us to define more sophisticated file structures. For example, a data set could be encapsulated in a task. The task will provide entries for reading and writing of its data. In this way, sharing of data does not create timing problems, also, special access rights may be checked in this way. Also, a particular installation may elect to create "dummy" users whose sole purpose is to provide shared data (corresponding to <subsys> on TOPS-20).

The user communicates with the machine via a uniform INTERFACE. All programming

environment facilities are accessed through this interface.

Details of the operations provided by the INTERFACE are described in subsequent sections.

C) Interface, Basic Operation

The sophisticated user should be able to access the full power of the programming environment and architecture but this facility should not defer the novice from performing meaningful work on the environment shortly after first exposure. The INTERFACE should help the user develop more sophistication.

INTERFACE command language, if the terminology is appropriate, must be user friendly, and not intimidating. The consistency between the programming environment and Ada should make this possible within the framework of our design.

Ideally we would want to provide one system mode, teach, use, novice and experienced user. Eteach, you use it once and than can forget what you learned,(it is boring to use Eteach a second time) the entire INTERFACE should be a teaching tool of sorts that reinforces learning and leads the user to new plateaus of understanding and functionality.

The INTERFACE must appeal to the customers management, to sophisticated users and to "novice" users. Many of the customers software people will have little Ada experience, as such they will be apprehensive about Ada. The INTERFACE should be user friendly but should enable the sophisticated user to access the full power available with the R-1000. The programming environment's contribution to reducing the Life Cycle Cost of software and improving productivity, as well as the provision of some tools to support the management of large software development projects will appeal to managers.

The R1000 INTERFACE is the only way for the user to communicate with the system, it acts as a window in the system. The basic component of the Interface is a structure oriented display editor. Every piece of data is displayed according to its type. For some common types the system provides suitable display routines, for example, Ada text will always be displayed in a properly indented form, while for type TEXT normal sequential display applies.

The user may provide his own data types with suitable display routines. For example, a two dimensional array may be displayed in table form with suitable column headings being automatically inserted.

Analogously, special input routines may be associated with a data type. In the case of Ada programs this is a incremental parser. It allows for programs to contain syntactic stubs but otherwise enforces syntactic validity of all programs input to the system.

The most important aspect of this scheme is that a running program (e.g. and elaborated package) is considered a data type; it can be edited like any other data object. This feature in effect make the editor a debugger; not special language has to be learned -- to debug a program simply means to edit it.

Clearly, the "edit operation" available for running programs differ from those for static program text. The operation available are:

-- while editing a running program, the display shows the corresponding program text.
-- normal cursor movement commands are allowed
-- any Ada statement or expression which is semantically valid in the scope determined by the current cursor position may be typed by the user and will be executed.
-- a declaration may be deleted from the program if it is not referenced
-- a new declaration may be elaborated if this would not cause a name conflict.

Note, that these are meta operations that manipulate a running Ada program, they are thus outside the semantics of Ada. These are the only additional concepts

used in the R1000 INTERFACE. The programmer should consider these meta operation to occur "outside" time; they transform one legal Ada program into a differen legal Ada program.

Except for the meta operations mentioned above all functions of the INTERFACE can be explained in Ada semantics. Interface functions may be overloaded and redefined according to Ada rules.

After login, the Interface will display the visible part of the user package on the screen. Commands allow to switch back and forth between the body and the visible part of the user package. Using basic cursor movement commands the user may inspect any part of his package. The Interface will only display objects visible in the scope of the user package. Only objects within the scope of the user package can be altered. Thus, Ada visibility rules are enforced by the Interface and guarantee system integrity.

All operations available on conventional operating systems can easily be expressed as edit operations in the above sense.

D) Programming Environment Tools

The choice of an initial tool set and the additional tools we announce and provide as updates to the programming environment must be coupled to user demand, needs, and expectations. We must not try to provide too much and cause ourselves undue problems. The R1000 will be well ahead of its time, the key is to get a sound, reliable, responsive implementation on the market in 1983. We can, from there, using feedback from our initial customers, by reviewing the applications where the R1000 is most popular, and the customers most interested in volume buying, embellish the capabilities of the programming environment. This is a cost effective, low risk, high probability of success implementation philosophy, it will work well for a product as advanced, with respect to the alternatives, as the R1000. R and D "bang for the buck" will be maximized as we will know more about our customers utility and demand curves.

Extensibility, add tools and add knowledge about tools and how they interact with each other and with the user. In this manner the unsophisticated user may have less information about certain tools than the sophisticated user, this prevents the unsophisticated user from improperly and harmfully invoking a tool.

An interesting market will the host-target where the R-1000 is initially the host and any number of micros or minis the targets. The programming environment must support this type of development, as specified or discussed in the Stoneman (?). This does not appear, to me, to cause any unusual problems; it requires that cross compiler for certain target machines be provided.

Invocation of tools and user programs is completely done with the facilities provided by the Interface.

The facility of the editor to execute any Ada statement or expression in the scope pointed to by the cursor is the basic means for invoking tools.

In addition to the basic method of program invocation other methods exist. Certain tools may be invoked automatically by individual editor commands. For example, whenever an Ada program is being edited, semantic analysis will happen automatically without the users knowledge. This feature has two main advantages. Large parts of the compilation time will be distributed evenly during the user's think time. Whenever the user asks for semantic analysis of his program, the list of current semantics errors is readily available. The conventional edit-compile-edit cycle of conventional systems does not exist on an R1000.

Obviously, the facility to customize and redefine Interface commands allows the user to invoke arbitrary sequences of tools with a single keystroke.

E) Teaching and Documentation

The R1000 will offer an excellent facility for teaching Ada, using Ada for documentation, an excellent Ada compiler, a good Ada editor, it will offer simple to use yet powerful on-line interactive tools, and will Price performance wise be extremely efficient. A customer may initially want an R1000 as the only cost effective system capable of teaching a large body of people Ada. From the teaching sub-market as a starting point we will generate repeat sales.

The environment should allow users to recognize the advantages of the R-1000 almost from their first exposure to the system. The user should become an advocate of the system to his/her management.

We will have sophisticated online documentation and teaching facilities. These too should appeal to a wide range of users; the novice should be able to learn to use the system with only a minimum of off-line instruction. At the same time the online features should be useful for the experienced user. I imagine that the programming environment will on request display comments and documentation; reading the Ada code itself should do the rest.

I expect that inexperienced programmers (e.g. managers who used to program) can write and execute Ada programs after a short (say 1 hour) time at a terminal. This time includes learning the system and learning ada.