* Notes for revision [illegible] with

```
+---------------+
! d i g i t a l !   I n t e r o f f i c e   M e m o r a n d u m
+---------------+
```

To: List

Date: 21 September 1987
From: Chip Nylander
Dept: Technical Languages

Subj: Thoughts on MICA and ULTRIX Software Architecture

The starting point for discussion of the relationship between MICA and ULTRIX software architecture and products is the question

What are we trying to accomplish?

The answers are (at least)

1. Provide high quality MICA and ULTRIX software systems.

2. Provide an ULTRIX program development and execution environment that will attract significant business to PRISM ULTRIX (meet the expectations of the ULTRIX market).

3. Get as much mileage as possible out of Digital's investment in software technology by applying as much of it as appropriate to the ULTRIX system.

4. Avoid pointlessly redundant software development efforts.

5. Lay the groundwork for applications that are portable and interoperable across multiple Digital systems.

6. Don't establish a software development bottleneck; in particular, don't impact the MICA software program.

7. Don't introduce unmanageable complexity.

8. Do something realistic.

This memo contains some of my thoughts on the matter.

I must hasten to emphasize that these are my opinions. This has not been widely reviewed in SDT, and cannot be taken as an "official" SDT position or as any sort of commitment by the SDT organization. (Wide review was not possible in the short amount of time I had available).

In addition, the commitment and level of available resources in SDT for development of PRISM ULTRIX layered products HAS NOT BEEN DEFINED. Until this is defined, we can only discuss these issues in terms of "what would be a good idea".

The following sections describe the components of software architecture that are relevant to layered products and our thoughts regarding those components, and my thoughts on run-time libraries, languages, and tools.

1 GOALS AND POSSIBLE SCENARIOS

We prefer the term "coordinated software architecture" to "common software architecture". While we believe (see below) that most of the software architecture between MICA and ULTRIX could be common and compatible, it is by no means proven that there can be 100% compatibility at all levels. There are some problem areas where we don't have all the answers yet. We prefer to recognize this fact by use of the term "coordinated".

X

Given potential goals for coordinated software architecture and common layered products across MICA and ULTRIX, the cross-product of all possibilities gives four possible scenarios:

1. An uncoordinated software architecture with non-common layered products.

   What this really means is "two completely seperate software development efforts for MICA and ULTRIX".

   Such a scenario might result in two sets of successful products, and is technically feasible.

   It is likely to result in poor compatibility, portability, and interoperablity between MICA and ULTRIX, and is likely to cost a great deal more (especially over time).

2. An uncoordinated software architecture with common layered products.

   This is the situation we have with VAX FORTRAN now.

   This is not a viable long-term scenario for a quality system. The result is a poor fit between some common layered product and one or both systems as the layered products base their design on the conventions of one system. The existence of two linkers on VAX ULTRIX, the JBL jacket-builder, inability to profile VAX FORTRAN programs on ULTRIX, etc. illustrate this scenario.

   Because it puts the layered products for both systems on one critical path, the schedule for one system might be at risk.

3. A coordinated software architecture with non-common layered products

What this really means is having just one architectural design and control group, but having parallel development of some layered products when required to meet the functional or schedule requirements of both systems.

For some layered products this might make sense. It would require coordination between layered product groups to insure compatibility, but might simply be required as an engineering tactic to get everything where it needs to be when it needs to be there.

This should only be done if really required for schedule reasons, or to avoid problems like introducing too much complexity or establishing a software development bottleneck.

4. A coordinated software architecture with common layered products

   This is the best of all possible worlds if we can pull it off and accomplish the list of goals on page 1.

   It would require careful architecture and product design.

   Because it puts the layered products for both systems on one critical path, the schedule for one system might be at risk.

There is an additional consideration: the MICA software architecture has been carefully designed for portability to an eventual 64-bit environment. Some additional uncoordinated software architecture may not have this property, and therefore have great problems getting to 64 bits in the future.

From these factors it is clear that the best approach is a coordinated software architecture. Common layered products represent an opportunity where that can be done without putting the schedule requirements of one or both systems at risk, degrading quality, or establishing a software development bottleneck. Non-common layered products may be required otherwise.

(Note that, since the commitment and level of available resources in sdt for development of PRISM ULTRIX layered products has not been defined, we can only talk about common layered products in terms of "what would be a good idea").

2 COMPONENTS OF SOFTWARE ARCHITECTURE

The components of the software architecture that are relevant to layered products include

1. Calling Standard -- entry descriptors, calling and return sequences, argument passing, call frame stack structure, stack usage, register usage, etc.

2. Condition Handling -- finding condition handlers, unwinding, capabilities of condition handlers, etc.
3. Status codes
4. Messages
5. Name space and naming conventions
6. Object language, object modules, and image files
7. Compiler -> Debugger interface (Debug Symbol Table)
8. Compiler -> Performance Collector Interface (Profiling Table)
9. Command Language interface
10. Services for managment of multithread execution
11. Remote Procedure Calls
12. Common Run Time Interfaces -- language, math, utility
13. IPSE
14. I/O and record management
15. Base System Services

There is some overlap here.

2.1 Calling Standard

There should be a common calling standard between PRISM MICA and ULTRIX.

There is no technical reason to do otherwise. The payoff is high for keeping the rest of the software architecture (and certain products sensitive to the calling standard) as common as possible.

Many of our RISC competitors use system-specific calling standards.

With the exception of C, programs written in high level languages do not depend on the calling standard, and such uses of C (e.g. VARARGS) can be caught by the compiler and handled.

2.2 Condition Handling

It seems inevitable that every system will provide system-specific primitive condition handling; vectored handlers on MICA, signals on

ULTRIX, etc.

Layered on this primitive handling should be a system-independent condition handling architecture that is oriented to the needs of languages and layered products. It should be stack-based, and should co-exist in some well- defined fashion with the primitive facilities provided by the system.

Capabilities should include

1. Static establishment of stack-based condition handlers

2. Raising continuable and non-continuable conditions, with arguments.

3. Access to condition codes and condition-specific arguments

4. Ability for a handler to pass on a condition with no action

5. Termination of a condition and resumption of program execution

6. Modification of a condition or its arguments

7. Addition of a subordinate or superordinate condition with arguments.

2.3 Status Codes

Status codes are somewhat problematical. Many VMS programs currently depend on the numerical encoding of status codes, as do many ULTRIX programs, particularly such system-specific properties of status codes as "low bit means success" (VMS) and "zero means success" (ULTRIX).

It may not be possible to have consistent status codes across MICA and ULTRIX without giving up VMS and/or UNIX compatibility.

2.4 Messages

The content of messages emitted by layered products should be common across MICA and ULTRIX. There is no reason to do otherwise. Perhaps the form of messages should be system-specific, and provide the "look and feel" appropriate for the host system.

Common message definition, formatting, and reporting interfaces should therefore be architected for both systems. The system-specific implementation of these interfaces should provide the "look and feel" required on the host system.

2.5 Name Space And Naming Conventions

Name space issues are somewhat problematical. VMS programs depend on case-insensitivity, and ULTRIX programs may depend on case-sensitivity.

In addition, many ULTRIX languages establish their own name space by adding language-specific patterns of underscores to all external names generated by the compiler. ULTRIX C, for example, prefixes all external names with an underscore; ULTRIX F77 both prefixes and postfixes all external names with an underscore.

This was done for reasons that should not exist in PRISM, namely to avoid collisions in the assembler between C internal compiler symbols and C user symbols, to avoid collisions in the linker between FORTRAN external symbols and C RTL symbols, etc.

However, there are ULTRIX C programs that have underscores explicitly appended by the programmer because the programmer knows that a FORTRAN symbol is being referenced.

We would have to deal with issues such as case sensitivity and explicit underscores in a coordinated software architecture.

2.6 Object Language, Object Modules, And Image Files, + object library

Here "object language" means the literals and linker commands that define the contents of the text and code segments; "object module" means the outermost envelope that encloses the object language and defines the structure of the module; "image file" refers to an executable entity.

There should be a common object language between PRISM MICA and ULTRIX.

There is no technical reason to do otherwise. Few, if any, ULTRIX programs depend on the internal details of the object language.

In addition, there is agreement in TL&E that the ULTRIX object language does not meet the need of modern languages such as ADA, so it would have to be extended anyway.

Some ULTRIX utilities depend on the structure of the object module. The cost of modifying these ULTRIX utilities is far less than the cost of introducing an incompatible object module format. (We also have to consider whether user programs depend on the ULTRIX object module structure; we don't know one way or the other).

The two systems may require system-specific features in the image file; however, there is no known reason why the overall structure should not be common.

Object Libraries Should be the same.

2.7 Compiler -> Debugger Interface (Debug Symbol Table)

There should be a common Debug Symbol Table definition between MICA and ULTRIX. There is no technical reason to do otherwise.

2.8 Compiler -> Performance Collector Interface (Profiling Table)

There should be a common Profiling Table definition between MICA and ULTRIX. There is no technical reason to do otherwise.

2.9 Command Language Interface

While command language interfaces are NOT one of the more pervasive elements of the software architecture, the layered products would benefit from the architecting of a common command line parsing interface between MICA and ULTRIX, similar to the CLI$ interfaces but with extensions as necessary to support ULTRIX command lines and DECwindows. Command termination status codes.

2.10 Remote Procedure Calls

The Digital RPC Architecture, when defined, should be compatibly implemented on MICA and ULTRIX.

2.11 Services For Managment Of Multithread Execution

The Common Multithread Architecture being defined by SDT should be the basis for management of multithread execution on both MICA and ULTRIX. These interfaces should be implemented on both systems.

2.12 Common Run Time Interfaces -- Language, Math, Utility

1. Language RTLs

   There should be common language RTL interfaces between MICA and ULTRIX. There is no technical reason to do otherwise. The feasibility of this has been demonstrated by prior language migration from VMS to ULTRIX.

   There may be system-specific entry points in each language RTL, but these should be provided only to support language features that are specific to that system -- common functionality should be provided by common RTL interfaces.

2. Math RTL

   The comments about language RTLs apply equally to the math RTL. (Note that ULTRIX-compatible interfaces to math functions could be layered on the common math function interfaces if required).

3. Utility RTL

   There should be common utility RTL interfaces between MICA and ULTRIX.

   In the fullness of time and corporate strategy, this component would be part of the Application Integration Architecture.

   Its contents may include common condition handling, command language parsing interfaces, remote procedure calls interfaces, parallel thread management interfaces, file system interfaces, memory and resource management, date/time services, string handling and translation, internationalization aids, data conversions, multiprecision arithmetic, execution statistics gathering, system information interfaces, and screen management.

2.13 IPSE

One of the goals of the IPSE project is to architect a set of callable and database interfaces that can be used by compilers and tools for coordinated, integrated support of the entire program development and maintenence cycle.

It is a specific goal for these interfaces to be implementable on a variety of hardware and software systems, to be usable by third-party tool developers to enhance the value of the Digital programming environment, and to be usable by enhanced existing tools.

The software architecture of MICA and ULTRIX should eventually include the IPSE interfaces (although these interfaces are still in a very early stage of design).

2.14 I/O And Record Management

As noted above, the language RTLs should provide a common I/O interface to the languages.

In addition, the utility RTL should architect a basic common interface to the host file system. This would include services to create, delete, and rename files, and to create, manipulate, and delete directories.

We believe that there should also be a common (AIA-conformant?) record management interface across MICA and ULTRIX (and VMS). However, this is not part of the RTL! We believe that architecture and implementation of a common record management interface is the responsibility of the operating system group(s). RTL-level I/O functions should be layered on this common interface.

2.15 Base System Services

It is inevitable that some layered products, to a greater or lesser degree, must utilize base system services that are specific to the host system.

It should be a goal of common languages, RTLs, and tools to use such services only when they are not available in the coordinated software architecture, and to consolidate such usage in as few places as possible.

3 LAYERED SOFTWARE PRODUCTS

3.1 Run Time Libraries

3.1.1 Language RTLs -

- o Definition of Interfaces and Capabilities

  The interfaces and capabilities provided by the language RTLs are private and established by agreement between the RTLs and the compilers.

  Where the same language support is being provided on multiple systems, the interfaces should be compatible.

- o Dependence on Software Architecture

  The language RTLs are most affected by condition handling, status codes, messages, services for management of multithread execution, I/O, and record management.

- o Dependence on Base System Services

  The language RTLs depend heavily on base system services -- their role is to provide a compatible implementation of language support on multiple systems.

  With sufficient investment, these dependencies could be isolated, but this would be a first-time investment which would likely be very significant.

- Reasonable approaches to FRS

  Unknown at this time. We have not established that such a language RTL could be delivered in time for PRISM ULTRIX FRS.

- Long-term strategy

  Unknown. We have not established that the language RTLs can be realistically targeted to all potential target systems without introducing unmangagable complexity, degrading quality, or establishing a software development bottleneck.

  To some degree, this depends on the compatibility of the system under the language RTL, which is currently unknown.

3.1.2 Math RTL -

- Dependence on Software Architecture

  Small dependence. There is some dependence on calling standard, condition handling, and status codes.

- Dependence on Base System Services

  There is only small dependence on the base system services.

- Reasonable approaches to FRS

  Unknown at this time. While a common math RTL appears the be the right long-term approach, we have not established that this can be delivered in time for PRISM ULTRIX.

- Long-term strategy

  A common math RTL seems a realistic long-term strategy, but only if this can be accomplished without establishing a software development bottleneck.

3.1.3 Utility RTL -

- Definition of Interfaces and Capabilities

  A portable utility RTL should be based on a new set of interface definitions, oriented towards the requriements of the Application Integration Architecture.

- Dependence on Software Architecture

The utility RTL depends most heavily on condition handling, status codes, messages, services for management of multithread execution, remote procedure calls (to provide functions for distribution of applications), I/O, and record management.

- o Dependence on Base System Services

  There is heavy dependence on base system services in the current VAX/VMS utility RTL. A new AIA utility RTL would be designed to isolate such dependencies.

- o Reasonable approaches to FRS

  Unknown. We have not established that this RTL could be delivered in time for PRISM ULTRIX FRS.

- o Long-term strategy

  A common AIA utility RTL seems to be the right long-term approach, subject to the usual caveats about what can be realistically targeted to all potential target systems without introducing unmangagable complexity, degrading quality, or establishing a software development bottleneck.

3.2 Compilers

3.2.1 C -

- o Definition of Interfaces and Capabilities

  In order to have a common implementation of C, it will be necessary to agree on a language definition.

  The usual language definition specified by UEG is "pcc plus system programming extensions for ULTRIX".

  The usual language definition specified by others is "ANSI C plus portable system programming extensions".

  There might also have to be system-specific application programming extensions, such as dictionary support for some system(s), source language extensions to integrate with the ported ULTRIX tools, etc.

  In any case, compatibility of external data representation (especially record structures) is important to allow data interchange between MICA and ULTRIX.

- o Dependence on Software Architecture

Calling standard, messages, name space and naming conventions, object language, debug symbol table, profiling table (and generation of profiling code where required), and command language interface would most affect a C compiler.

We would expect the C run time library to be the standard ULTRIX library.

- o Dependence on Base System Services

  Compilers are not much dependent on base system services, and such dependencies can be localized.

- o Approaches to FRS

  It seems realistic to develop a common C compiler for MICA and ULTRIX; it is also clearly the best approach to getting consistently high quality compilers on both MICA and ULTRIX, to get more mileage out of our optimization technology, and to insure the right level of C language compatibility across MICA and ULTRIX.

  We really should not write two C compilers for PRISM if there's any way we can avoid it.

  Unfortunately, this puts one C compiler group on the critical path for both systems. PRISM ULTRIX is scheduled to ship first, and supporting field test and release processes is NOT free -- in fact, it gets harder all the time. There is a risk that getting involved with PRISM ULTRIX will put the MICA schedule at risk.

- o Long-term strategy

  Do once and keep it common.

3.2.2 FORTRAN -

- o Definition of Interfaces and Capabilities

  The FORTRAN language definition should be VAX FORTRAN with appropriate system-specific extensions, including intersystem compatibility-flagging on each system. Compatibility of external data representation (especially record structures) is important to allow data interchange between MICA and ULTRIX.

- o Dependence on Software Architecture

  Calling standard, messages, name space and naming conventions, object language, debug symbol table, profiling table (and generation of profiling code where required), command language interface, multithread services (eventually), and common run time interfaces would most affect a FORTRAN compiler.

- o Dependence on Base System Services

  Compilers are not much dependent on base system services, and such dependencies can be localized.

- o Approaches to FRS

  It is realistic to develop a common FORTRAN compiler for MICA and ULTRIX; it is also clearly the best approach to getting consistently high quality compilers on both MICA and ULTRIX, to get more mileage out of our optimization technology, and to insure the right level of FORTRAN language compatibility across MICA and ULTRIX.

  There should be a common FORTRAN compiler.

  Unfortunately, this puts one FORTRAN compiler group on the critical path for both systems. PRISM ULTRIX is scheduled to ship first, and supporting field test and product release will not come for free. There is a risk that getting involved with PRISM ULTRIX will put the MICA schedule at risk or (more likely) may result in a less agressive FORTRAN product for MICA FRS.

- o Long-term strategy

  Do once and keep it common.

3.2.3 Other Languages -

- o Definition of Interfaces and Capabilities

  There should be a common language definition across all DEC systems -- VAX/VMS, VAX/ULTRIX, PRISM/MICA, and PRISM/ULTRIX -- except where system-specific extensions are appropriate. The language definition will normally be based on the relevant standard. We should plan to support compatibility-flagging in all compilers to prevent applications from accidentally depending on system-specific extensions.

- o Dependence on Software Architecture

  Same as FORTRAN -- most compilers will depend mostly on calling standard, messages, name space and naming conventions, object language, debug symbol table, profiling table (and generation of profiling code where required), command language interface, multithread services (eventually), and common run time interfaces.

- o Dependence on Base System Services

  Compilers are not much dependent on base system services, and such dependencies can be localized.

- o Approaches to FRS

  We don't anticipate any other languages for PRISM ULTRIX FRS.

- o Long-term strategy

  Do once and keep it common, subject to the usual caveats about introducing unmangagable complexity, degrading quality, or establishing a software development bottleneck.

3.3 Tools

3.3.1 Debugger -

- o Definition of Interfaces and Capabilities

  The traditional ULTRIX debugger uses the "stab" interface between compilers and the debugger.

  We know of no technical reason why the new Debug Symbol Table is not an adequate interface for FORTRAN and C, and we believe that it is the only right interface for future debugging requirements such as vectorization, multitasking, and ADA.

  The emphasis on future debugger human interfaces should be on DECwindows, with as compatible of an interface across systems as possible. The command interface should be de-emphasised.

  However, where there is a command interface, the right one seems to be a dbx-like interface.

- o Dependence on Software Architecture

  The debugger depends on most of the software architecture: calling standard, condition handling, status codes, messages, name space and naming conventions, image files, debug symbol table, command language interface, multithread services, remote procedure calls, run time interfaces, IPSE, I/O and record management.

- o Dependence on Base System Services

The debug kernel is highly dependent on the base system services and the hardware architecture.

- Approaches to FRS

  We have no realistic plan to get any portion of the debugger being built for PRISM MICA ready for PRISM ULTRIX FRS.

  The only approach we know that might produce a debugger on the required schedule is to port dbx to PRISM ULTRIX, converting it to use the new debug symbol table.

- Long-term strategy

  Unknown.

  Common debug technology is attractive -- it consolidates investment in new debug technology such as support for vectors, parallel processing, debugging optimized code, and complex languages such as ADA.

  However, we have not established that the debugger can be realistically targeted to all potential target systems without introducing unmangagable complexity, degrading quality, or establishing a software development bottleneck.

3.3.2 IPSE -

- Definition of Interfaces and Capabilities

  One of the goals of the IPSE project is to architect a set of callable and database interfaces that can be used by compilers and tools for coordinated, integrated support of the entire program development and maintenence cycle.

  IPSE will control its program and database interfaces, but they will be designed for portability to multiple system and for support of a wide spectrum of tools.

  Tools providing human and compiler interfaces will be built on top of this IPSE platform. We expect this to include existing VAXset tools, third party ISV-supplied tools (IPSE will be an open architecture), and possibly existing ULTRIX tools.

- Dependence on Software Architecture

  IPSE will be designed to be as independent of software architecture as possible. The component of the software architecture that will most affect IPSE will be messages and status codes, utility RTL interfaces, record management, and possibly remote procedure calls (for distributed functions).

o Dependence on Base System Services

  IPSE will be designed and implemented to cope with different base system services.

o Approaches to FRS

  IPSE cannot be there for PRISM ULTRIX FRS. The first FRS of IPSE will be on VAX/VMS. The strategy beyond that point is not yet defined.

o Long-term strategy

  Provide IPSE compatibly on all appropriate systems.

3.3.3 Other Tools -

o Definition of Interfaces and Capabilities

  This is the $64 question. There is no clean statement of requirements for tools on ULTRIX. Providing VAXset on ULTRIX is frequently mentioned as a wish-list item, but there is no consistent definition of the meaning of this.

  - Are such tools required for programmer portability between VMS and ULTRIX, or to compete with third party tool products, or because the tools available on ULTRIX are inadequate?

  - Is the human interface style of existing tools (such as the DCL-style command syntax of LSE) acceptable, or does the LSE command interface, for example, have to be ultrixized?

  - Do tools ported from VMS (such as CMS) have to integrate with the native ULTRIX tools (such as SCCS)?

  - etc.

  We don't know the answers to these questions. Until someone makes some believable and consistent statements about tools requirements for ULTRIX, we won't know the answers.

  We CAN state that future user interfaces should be based on DECwindows, and that command-line interfaces should be de-emphasized.

o Dependence on Software Architecture

The software architecture that affects this class of tools is messages, name space and naming conventions, command language interfaces, the utility RTL, IPSE, and record management.

o Dependence on Base System Services

  Most of the existing VAXset tools are closely tied to VMS. It would be a major ripup to bring them to ULTRIX.

o Approaches to FRS

  Unknown -- we need to understand the requirements first.

  An additional possibility is to provide VMS-based tool servers for functions such as CMS, providing only human interfaces on the PRISM ULTRIX system.

o Long-term strategy

  The long-term strategy should be to get IPSE on ULTRIX and orchestrate the right tools functionality (whatever that is) across native ULTRIX tools, ported VAXset tools, and third party ISV-supplied tools, all built integrated on the IPSE platform.

  Whether there can or should be common tools is not yet established. Without understanding the requirements, and without the right underlying common base of software architecture, IPSE, DECwindows, etc. this cannot be asserted with any confidence.

4 SUMMARY

We believe we understand what the key components of a coordinated software architecture are. We know of no technical reason why there should not be a coordinated software architecture between PRISM MICA and PRISM ULTRIX.

Among the benefits of such a coordinated architecture would be opportunities for common layered products.

Whether any common layered products could be developed in time for PRISM ULTRIX FRS is not a subject of this memo. Those requirements and commitments have not been defined. We are, however, concerned about the impact of having many layered products on the critical path for both PRISM ULTRIX and PRISM MICA (which both have aggressive schedule goals).

Compilers, IPSE, an AIA utility RTL, and the math RTL appear to be reasonable risks for a long-term common product strategy if there is a coordinated software architecture in the areas that they depend on.

We have not established that language RTLs and tools (including debuggers) are reasonable risks for a long-term common product strategy. There is danger of missing important quality and schedule goals if we try to do too much here.

digital™ **INTEROFFICE MEMORANDUM**

TO:	Distribution	DATE:	March 24, 1988
		FROM:	Tom Miller Steve Jenness Mark Ozur Jim Jackson
		DEPT:	DECwest Engineering
		LOC:	ZSO
		ENET:	DECWET::

SUBJECT: Mica Working Design Document Overviews

The Mica WDD

The DECwest Mica Software Development and Technical Writing teams are proud to distribute the *Mica Working Design Document Chapter Overviews*. These Overviews are a by-product of the design process for Mica, as described later in this memo.

Mica is the proprietary operating system for PRISM architecture machines, and a new member of the DIGITAL/VMS computing environment. It is the base system software for the Cheyenne database server and Glacier compute server.

Mica is a symmetrical multiprocessing (SMP), multithreaded operating system with a number of features to promote modular growth. These features include an executive object architecture, layered I/O system, protected subsystems support, and remote procedure call (RPC). In addition, a powerful set of client/server mechanisms have been designed to support the initial server-based FRS products.

This document contains the collected chapter overviews of the Mica Working Design Document (WDD). The WDD is both a functional specification and a design specification. You should read this document to gain a basic understanding of Mica and its various components at an overview level, since Mica will be the basis of many PRISM-based products in the future.

Three actual chapters have been included in the *Mica Working Design Document Chapter Overviews* because they provide useful overviews of Mica and the initial FRS products. These chapters should be of particular interest to all readers. They are *Introduction to Mica* (Chapter 1), *Cheyenne Overview* (Chapter 48), and *Glacier Overview* (Chapter 50).

The distribution list for this document was formed, in part, from the distribution list of the first draft of the *PRISM Software Working Design Document*, with the addition of other senior consultant and key management personnel.

The remainder of this cover letter contains a description of the design process used to develop the Mica WDD, including the overview process, the chapter process, and the current status of the project. This description is not required for an understanding of the Overviews Document, but is presented here for your information, to show how the production of the chapter overviews fits into the larger Mica design process.

The Design Process

The sheer size of the Mica project, in terms of both the magnitude of required functionality and the number of people involved, demands a rigorous design process. All aspects of the system have to be carefully designed and specified, including functional requirements, partitioning into components, component interfaces, and the internal design of each component. The specifications are organized into 58 chapters, with each chapter assigned to one of 38 engineers. Each engineer is assigned to one of 12 technical writers for assistance in producing the chapter.

The design review process is coordinated by three software architects. It is divided into two major steps: the production of a preliminary chapter overview, and the production of a design chapter. The design chapter generally begins with a final version of the chapter overview. Most of the preliminary chapter overviews range in size from three to six pages.

The Overviews

There are a number of good reasons to start with an overview of the chapter:

1. Most importantly, it forces the engineer to take a high-level look at the requirements, identify his fundamental approach, and then capture these things in a short paper before getting lost in low-level details.
2. The overview then serves to force discussion at an early stage before the responsible engineer feels committed to a detailed design. At this point in time, it is still possible to make significant changes in the design or, in an extreme case, start over again.
3. The overview serves as an early communication mechanism within DECwest for other engineers, product managers, technical writers, and CSSE to gain a basic understanding of the various system components. It serves the same purpose for other groups outside of DECwest with Glacier or Cheyenne FRS deliverables. Collected in this *Mica Working Design Document Chapter Overviews* document, the overviews are now a communication mechanism for use within the entire corporation.
4. Finally, completion of the overviews allows the detailed design chapters to be written in parallel, in the same manner that the completion of the detailed design subsequently allows the implementation to proceed in parallel.

The Overview Process

For the review of each overview and subsequent chapter, the responsible engineer chooses a primary review group from among the other engineers in his group, his project leader, supervisor, and potentially anyone else who has to interface with the component or has other concerns about the design. The last category frequently includes representatives from groups outside of DECwest.

The steps in the preliminary overview process proceed as follows:

1. The preliminary overview is written by the engineer, with assistance from his technical writer.
2. The overview is then reviewed by the primary review group and revised by the engineer.
3. Next, the overview is reviewed by the software architects and revised by the engineer. In addition to having normal review comments, the architects are responsible for consistency and completeness of the design across the whole system.
4. The overview is then posted in a notefile for general review and discussion.

5. After allowing time for the more critical design issues to be raised and resolved via replies in the notefile, the overview discussion period is officially closed. This closure is necessary to make people aware that they only have a finite time period during which they can raise substantive issues.

The Chapter Process

After completing the overview, the engineer proceeds to write the detailed design chapter, with the assistance of a technical writer during and/or after the first draft. Generally, the production of the chapters has proceeded quite efficiently, due greatly to the fact that much of the indecision and controversy has been dealt with already in the overview process. Chapter sizes are ranging from a dozen pages to over 100 pages.

Chapter review proceeds much like overview review, as follows:

1. Once written, the chapter is reviewed by a primary review group.
2. The chapter is then reviewed by the software architect responsible for that part of the project.
3. The chapter is posted in the notefile. Once the chapter is posted, it goes under ECO control.
4. Some time after posting of the chapter, a presentation is given on the chapter, possibly in conjunction with other chapters.

Status of the Overviews

Although the previously described process is helping us to design Mica in an orderly fashion, it is still important to point out that some changes are inevitable. The overviews are only guaranteed to capture a snapshot of the design. We are confident, however, that the overviews do present a good overall picture of Mica, which will not change in any fundamental way.

The *Mica Working Design Document Chapter Overviews* contains a mixture of preliminary chapter overviews and final versions of the chapter overviews that have been revised during development of the chapter. Clearly, with the presentation of major parts of the system being limited to a few pages each, the level of detail is also limited. Much of the detail is naturally left to the final design chapters.

It is also important to point out that neither the overviews nor the chapters themselves are intended to portray which features will actually be present in either of the FRS products. This is the purpose of the Phase 1 documentation.

Status of the Mica Project

Design chapter work is nearly complete. Early implementation work is now in progress and on schedule. The current baselevel contains a special development environment, the kernel, and a preliminary executive with threads, context switching, condition handling, basic synchronization objects, and a minimal I/O system (no device support yet). Coding is now proceeding in most areas of the system.

From: TLE::MITCHELL "Charlie, TLE::MITCHELL 25-Mar-1987 1251" 25-MAR-1987 12:
To: DECWET::CUTLER,DECWET::MILLER,CLT::GREENWOOD,NYLANDER,CONTI,SMURPHY,MITC
Subj: Ada, threads, and priorities

We've reviewed Steve Greenwood's trip report on the discussions at DECWEST last week and felt that it was important to provide you with some feedback with regard to Ada's future implementation on MICA.

1. The Ada project is definitely planning on a one-to-one correspondence between Ada tasks and MICA threads. There will be ONE task per MICA thread.

2. As best we understand the proposed major/minor priority scheme, we feel that it is workable. We have not been able to come up with any better alternative. However, we hope that the implementation does not penalize Ada processes relative to non-Ada processes.

   Priorities in Ada are used to indicate the relative urgency of various tasks. Many applications depend on multiple priority levels and won't work as intended without them. We currently support 16 priority levels under both VMS and VAXELN. Although validation is possible with fewer than four Ada priority levels, we believe that we need at least four levels to meet customer expectations and to have a successful product.

From: TLE::MITCHELL "Charlie, TLE::MITCHELL 21-Apr-1987 1714" 21-APR-1987 17:
To: NYLANDER,CLT::GREENWOOD,GROVE,CONTI,SMURPHY,MITCHELL
Subj: Ada priorities and PRISM

I interchanged several mail messages with Dave Cutler regarding the Ada priority issue and sent the one below on 6 April. I have not heard from him since that time and thus assume that the issue has been closed. If you hear anything to the contrary, please let me know.

Charlie

This note summarizes my position on the Ada priority issue.

From my understanding of the current design of the scheduler, it appears that Ada multi-thread programs written for time-sharing use on PRISM will perform much better if users avoid specifying priorities. (This assumes that Ada recognizes this case and only sets the bit that inhibits priority boosts when absolutely necessary.)

Nonetheless, I believe that Ada needs the following to have a successful product on PRISM:

1. At least four minor priority levels

2. The bit that inhibits priority increments (unless the priorities of the threats in a process remain the same relative to each other)

Four priority levels seems adequate for most applications designed for time sharing use. If we supported fewer than four priority levels, the perception in the Ada community would be that "PRISM Ada doesn't support priorities." Even with a highly trained sales force (which we don't have), it would be very difficult to overcome such a perceived deficiency. Program portability and machine independence are very important issues in the Ada community. It's one thing to recommend that people avoid using priorities; it's another to force people to avoid them. Even with four priority levels, we can anticipate some resistance from some users who expect 16. However, the difference between 16 and 4 priority levels is less significant than the difference between 4 and 2.

Assuming, then, that minor priority levels are supported, we need the bit to inhibit priority boosts to satisfy the semantics of the language.

I hope that we can now close this issue. Thank you again for your clarifications.

Charlie

```
From:    TLE::DECWET::MILLER "Tom Miller, DECwest Engineering, (206) 865-8770  02
To:      TLE::NYLANDER,MILLER
Subj:    CRYSTAL/JEWEL/MICA Update
```

```
    +-+-+-+-+-+-+-+
    |D|I|G|I|T|A|L| I n t e r o f f i c e   M e m o r a n d u m
    +-+-+-+-+-+-+-+
```

```
    To: Chip Nylander                          From:       Tom Miller
                                               Dept.:      PRISM Software
                                               Mailstop:   ZSO
                                               Telephone:  206-865-8770
                                               Network:    DECWET::MILLER

                                               Date:       Mar. 2, 1987
```

Chip,

What the hell is MICA, you ask? Well, Dave threatened unspeakable retribution if we didn't come up with a different name for P.TBD, so I called a "name that system" meeting in our Mica (heh, heh) conference room. The progress of that meeting was about as pismal as the efforts to name the software have been to date; generally no one was real happy.

So I said, why don't we just name it MICA. We can say it stands for "Multiple-Interface, Concurrent Architecture", which are indeed the two major themes that all of us, including Dave, wanted to work into the name. This suggestion gained the support of the naming meeting, and we decided that for a code name, it would do. Reid is doing the legal checks for name conflicts, just the same.

Any way, I thought I would give you some scheduling updates. We are moving our Phase 1 two months, which will make it some time in August. As estimating and perting and WDD work goes on, we realize we really need that time, and the hardware group is not complaining about the change either.

More significantly, Rob has found it necessary to slip the FRS date for Jewel to August, 1989. We will now define the software functionality to meet this date. Naturally we are hoping that those components that SDT is delivering that we rely on will still be available as scheduled; which would ultimately give us all more time to beat on the system.

This change will allow us to put even more emphasis on testing with Emerald prototypes, which will be available in April 88 according to current schedules. We should be able to do a really good internal field test on Emerald now. In fact, from a software standpoint we should not preclude the possibility of a minimal functionality release on Emerald ahead of Jewel, if the development plan for Jewel makes that reasonable.

As MICA approaches Phase 1, we should exchange more detailed scheduling information as we have it. We will be using a pert chart tool called VUE. Since our project plan goes to the printer at the end of May, we should be able to give you some detailed scheduling information well before that.

PRISM Technical Interest distribution -- Page 1

Mike Anderson	ZKO2-3/N30
Brian Axtell	ZKO2-3/R56
Susan Azibert	ZKO2-3/Q08
Bertril Beander	ZKO2-3/N30
Hal Berenson	ZKO2-2/N59
Bill Bernson	ZKO1-1/M26
David Blickstein	ZKO2-3/N30
Mark Bramhall	ZKO2-3/R56
Ron Brender	ZKO2-3/N30
Walter Carrell	ZKO2-3/N30
Keith Comeford	ZKO2-3/R56
Scott Davis	ZKO2-3/Q08
Neil Faiman	ZKO2-3/N30
Jim Flatten	ZKO2-3/R56
Dan Frantz	ZKO2-3/R56
Liz Freburger	ZKO2-3/Q08
Peter Gilbert	ZKO2-3/K06
Steve Greenwood	ZKO2-3/K06
Rich Grove	ZKO2-3/N30
Kevin Harris	ZKO2-3/N30
Steve Hobbs	ZKO2-3/N30
Ken Hobday	ZKO2-3/K06
Jim Kapadia	ZKO2-3/Q08
Jim Kellerman	ZKO2-3/M31
Steve Klein	ZKO2-2/N59
Brian Koblenz	ZKO2-3/N30
Matt Lapine	ZKO2-3/K06
Tom Lavigne	ZKO2-3/K06
Glenn Lupton	ZKO2-3/N30
Charlie Mitchell	ZKO2-3/N30
Dave Moore	ZKO2-3/N30
Chris Nolan	ZKO2-3/N30
Bill Noyce	ZKO2-3/N30
Chip Nylander	ZKO2-3/N30
Jim Ravan	ZKO2-2/N59
John Reagan	ZKO2-3/N30
Tom Scarpelli	ZKO2-3/K06
Neil Schutzman	ZKO2-2/N59
Laura Schwartz	ZKO2-3/N30
Thomas Siebold	ZKO2-1/N71
Al Simons	ZKO2-3/K06
Joyce Spencer	ZKO2-3/N30
Barry Tannenbaum	ZKO2-3/R56
Sue Thorstensen	ZKO2-3/O04
Rich Title	ZKO2-3/N30
Jim Totton	ZKO2-3/K06
Stan Whitlock	ZKO2-3/N30
Jeff Wiener	ZKO2-3/K06
Tom Wimberg	ZKO2-2/M37
Paul Winalski	ZKO2-3/N30
Linda Zaharee	ZKO2-3/R56

APPENDIX A

CURRENT WDD CHAPTER ASSIGNMENTS

The following is a list, broken down by functional grouping, of the current chapter assignments. This list may not be complete and very likely will change over time.

Where an author's name appears in brackets, it is to serve as a place-holder until the chapter is assigned.

GENERAL

Architecture Overview and Introduction	Miller
Naming and Coding Standards	Schreiber
Status Codes & Messages	Ballenger
Type, record, etc. name appendix	(Perazzoli)

EXECUTIVE

Object Architecture	Perazzoli
Process Structure	Lucovsky
Kernel	Cutler
Memory Management	Perazzoli
I/O Architecture	Olivier
Condition & Exit Handling	Bismuth
System Service Architecture	Walker
Security & Privileges	Walker
Booting	Walker
System Services	(Perazzoli)
SDA and Kernel Mode Debugger	(Bismuth)
Auto System Crash Recovery	(Fries)

I/O AND FILE SYSTEM

Disk Function Processors	East
Diagnostics	Brown
Error Logging	Brown
Protected Subsystems & RPC	Ozur
Message Function Processor	Fries
Directory Structured Function Processors	Tyson
ODS Function Processor	Tyson
Caching	Brundrett
ANSII Magtape	(Bismuth)

RMS	Chatterjee
File Management Util. (Backup, init, etc.)	Brundrett
Console Support	Walp
IMAGE RELATED	
Object Module & Image File Format	Peterson
Image Activator	Perazzoli
RPC Stub Compiler	Lenzmeier
SYSTEM MANAGEMENT AND ADMINISTRATION	
Layered Products & System Disk	Walp
System Management	Girdler, Ditto
Software installation and Update	Ditto
Operator Communications	Girdler
TESTING AND QUALIFICATION	
Performance	Sestrap
Failure Modes & Effects Analysis	Schreiber
Testing	Schreiber
UETP	Looi
NETWORKS	
Network Architecture	Fries
Network Components (n chapters)	(Fries)
DECnet	Kelly
DATABASE SERVER	
Common Logging	Miller
Host DBM Communications	(East)
DBM Inter-box Communication	Wickham
CRDK IPC	Dunlap
Any other Database Specific Chapters	(East)
WORKGROUPS	
Workgroups (n chapters)	(Saether)
COMPUTE SERVER	
AIA (n chapters)	Connors
RPC Callback Libraries	Ozur
Host Side of Compute Server	Doherty

APPENDIX B

RESPONSIBILITIES LIST

The following list identifies individuals as responsible for specific parts of the system. The list is not yet complete and will evolve over time.

An asterisk after a name indicates the individual is a project leader.

Project Function or Element	Responsible Person	
I/O	Jeff East	*
Testing	Benn Schreiber	*
Performance	Kathy Sestrap	
Networks	Jim Kelly	*
RPC Transport	Kevin Dunlap	
File System	Joan Tyson	*
Workgroup Transport	Steve Jenness	*
RMS	Sumanta Chatterjee	*
Executive	Lou Perazzoli	*
Object Architecture	Jim Walker	
Process	Mark Lucovsky	
I/O Architecture	Charles Olivier	*
Development Environment	Dave Walp	*
Linker/Library	Kim Peterson	
AIA/DECwindows	Myles Connors	*
RPC	Mark Ozur	*
RPC Stub Compiler	Chuck Lenzmeier	
C-RTL	David Ballenger	*
Diagnostics	Richard Brown	*
Workgroups	Chris Saether	*
Booting	Jim Walker	
Client-Server Interface	Dave Ballenger	*
Fault Tolerance	(Perazzoli)	
Security	Jim Walker	
System Services	(Perazzoli)	
Caching	Peter Brundrett	
Backup/disk utilities	Peter Brundrett	
IPC	Kevin Dunlap	

The Application Integration Architecture (AIA) and Mica Compute Server: The AIA "Strawman"

Revision 1.0

14-September-1987

Author:

Myles F. Connors Jr.

Major Contributors:

Al Simons

Jeffrey C. Wiener

digital™

Revision History

Date	Revision Number	Summary of Changes
14-September-1987	1.0	Initial version

1 Overview

This document is designed to serve as a point of departure for further discussions on the role of AIA in the development of Mica-CS, the PRISM compute server software product being developed at DECwest.

This document is not meant to be a definitive statement of what is or is not contained in the FRS Mica-CS product. Rather, this is a first pass at describing the required capabilities of various AIA architectures in order to share my understanding about the relative complexity of the work involved in using AIA as the basis for the application program interface to Mica-CS. That understanding of the magnitude of the tasks required is for input to the Mica-CS project scheduling process.

This document also describes a personal vision of an eventual fully realized model of AIA on Mica. Some thoughts on how this vision might be achieved using a phased implementation are presented.

Finally, this document is designed to be the starting place for DECwest putting a "stake in the ground" about the role of AIA in future DEC software systems.

2 Some Common Misconceptions About AIA

Because AIA is not fully defined and understood, there are some common assumptions made about AIA that need to be examined.

2.1 AIA Routine Implementations Must Be Portable

With the list of AIA target operating system environments as large as it is (VAX/VMS, MICA, ULTRIX, MS-DOS), and the wide disparity of capabilities provided by those operating systems, it is hard to believe that a common portable solution for every AIA-level problem can be found.

The goal of AIA is that the *interface* to AIA routines be portable. The underlying code may not be "implementable" on every target operating system. It must be possible, however, to provide these interfaces on each of the target operating systems. The actual implementation of those routines are distributed via RPC or other distribution mechanism.

\These statements are easier to defend when talking about the "large granularity" AIA capabilities such as the Print System Model. It will be a challenge to provide AIA capabilities in facilities with interfaces that are of much finer granularity, e.g. process creation. These fine grain solutions will not be useful if the cost of distributing the interface grossly exceeds the inherent cost of the algorithm used to implement the capability.\

Having said the above, it is still highly desirable to be able to port the implementations of AIA routines from operating system to operating system, where reasonable. This should be a goal of any AIA software we develop at DECwest, if only to help guarantee that the capabilities that we need to be seamless with our client systems (presently VAX/VMS and VAX/ULTRIX) are indeed present on those systems at FRS.

\This seems to imply that we either port PILLAR to VAX/ULTRIX or we use C as the common implementation language for any new code developed specifically for AIA on Mica-CS.\

2.2 AIA Is All Things To All People

AIA is not a panacea. The original target for AIA is the ISV's that we can induce to port their applications software to our products, thereby leveraging our hardware sales.

This means that we need to target our earliest offerings at the low level "nuts-and-bolts" capabilities that will make AIA attractive to ISV's. The most obvious such capability is RPC; RPC is the cornerstone of AIA.

Later AIA offerings can be used to fill in the gaps that would appeal to customers writing code for limited audiences.

\It is for this reason that I recommend that the Corporation proceed cautiously before rushing to establish the various AIA architectures as external standards. Our energy would be better spent in delivering the capabilities in the short term to ISV's than in reducing the whole thing to pablum suitable for every possible end user.\

3 Compute Server Customers and Their Applications

I believe that the compute server as specified today will sell mainly into the traditional DEC markets of scientific and technical customers. The customer who heretofore was unable to afford the compute server resource he/she really wanted (for example, a Cray), will now be able to afford a Glacier system. Existing Digital customers will appreciate that this is a single-vendor solution to their problems.

Cray and other high-end suppliers will always be able to provide the highest absolute performance based on their willingness to use the most exotic and risky technology available to them. They are also less sensitive to producing systems on a piecemeal basis. Digital and other large manufacturers will invariably be held at bay based on the traditional requirements of having all implementations be based on more stable technology.

The introduction of Glacier will not change this.

To be able to compete successfully in this space and expand our markets we will need to draw upon our unique strengths in providing integrated, supported solutions to our customer's computing needs.

For Glacier, AIA can provide a path for Digital to distinguish Glacier from its competitors based on the portability of applications that AIA can make possible.

3.1 The Picture at FRS

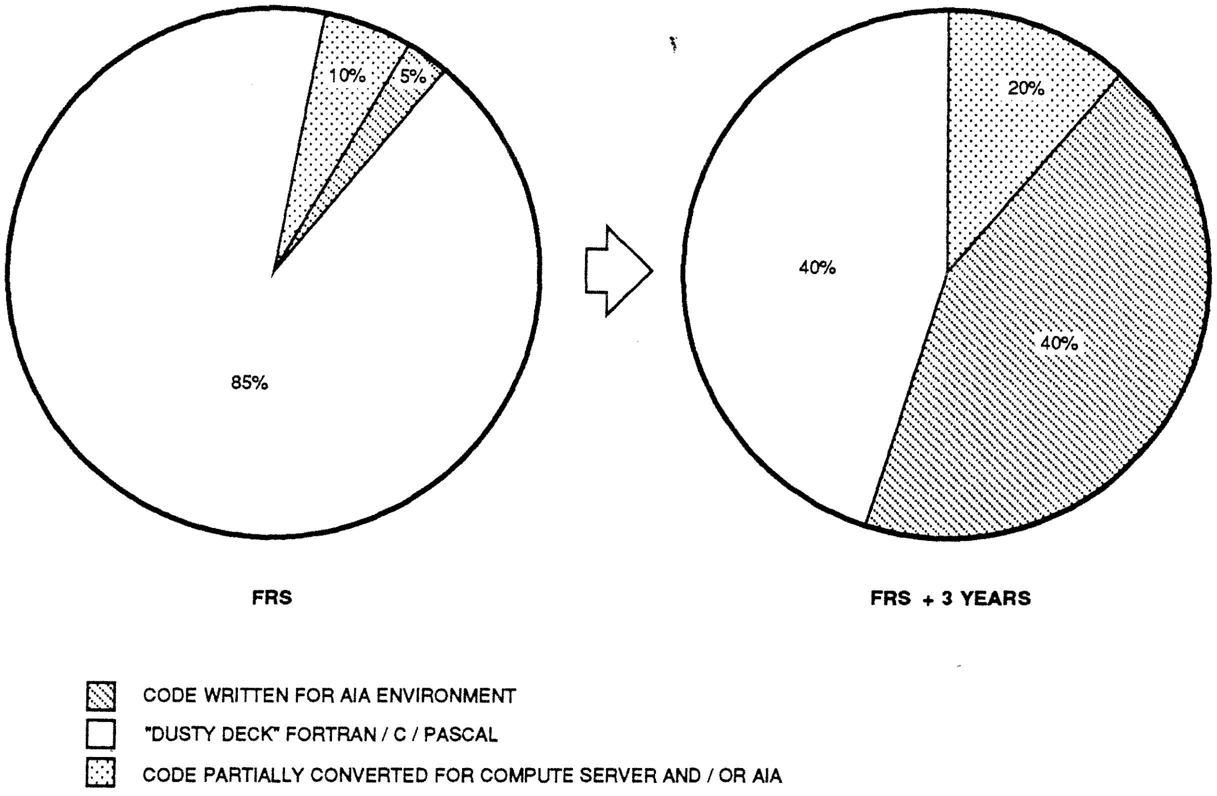
Figure 1 shows my estimate of the derivation of all of the code running on the compute server at FRS (roughly 1990) and approximately three years later. This includes both Digital-supplied and customer-supplied code.

An assumption made here is that AIA has achieved a foothold across VAX/VMS and VAX/ULTRIX in the three year timeframe after the compute server FRS. That is the result of an aggressive program of inducing ISV's to port or create applications based on AIA. Those applications are starting to appear in common use.

At FRS, most of the code running on the compute server is envisioned to be FORTRAN, C, or Pascal programs that implement some technical/scientific algorithm that is compute-intensive. More specifically, these are "high-headway" compute-intensive programs.

I use the word "program" instead of the word "application" purposely. These programs are typically handcrafted solutions to specific problems that have been begrudgingly ported by non-programmers to successive machines over the years.

Figure 1: Derivation of Code Running on the Compute Server



These programs typically invoke only those capabilities of the underlying operating system made visible through a language RTL. For that reason, these programs are largely portable already. Answers to the questions related to porting these types of programs are addressed in the compiler and language RTL documentation provided with the associated compiler products.

The presence of AIA on Mica-CS is largely uninteresting to this class of users. Seamlessly developing programs on the compute server client and recompiling/relinking them to run on the compute server will be attractive to these users. The applicable dimension of "seamless" here is that there are implementations of the language RTLs on both the client and server systems.

Those customers who can recognize the long-term advantages of AIA *and who are willing to modify their code to take advantage of these capabilities* are represented by the 10 percent slice of the pie chart on the left in Figure 1.

The message from the EIP trip reports seems to be a universal one of "don't make us change anything unless you can show us immediate, large benefits." Even if Digital embarked on a crash course in establishing AIA today, it is hard to imagine that there would be more than a limited presence of AIA in the FRS timeframe. More importantly, as our DECnet experience has shown, the lag time of customer acknowledgement of the benefits of AIA will push the *perception of an effective AIA presence* out further in time, despite the products we may ship today.

The remaining 5 percent slice of the pie chart on the left of Figure 1 represents code shipped by Digital or produced by selective ISV's through inducements by DEC. The greatest challenge for DECwest software marketing will be to guarantee that this percentage will be higher at FRS through an aggressive program of selecting and supporting ISV's with high-visibility and high-volume applications.

\My personal opinion is that having high-visibility applications present at FRS is as important as having high-volume applications. Until we have a program of compute servers that spans the "procedure to batch-job" spectrum, I assume that Digital-supplied compute servers will not achieve the volume of our traditional processor products; they will remain somewhat specialized solutions. Having high-visibility applications present early lends credibility to AIA and reduces the "time to customer appreciation" curve.\

This is a long range problem and requires a long-term commitment from the Corporation to ensure success. We have to be prepared to promote our approach in a consistent and understandable fashion from the start or risk the fate of DECnet: Digital's long-term commitment to DECnet is just recently being reflected in the customer loyalty and the general perception of the completeness of the solution.

3.2 The Picture at FRS + 3 Years

The right hand pie chart in Figure 1 shows a modest growth in the amount of code modified to run on the compute server and/or AIA. The portion of the code written expressly for AIA has increased, but is still matched by the volume of "dusty deck" code.

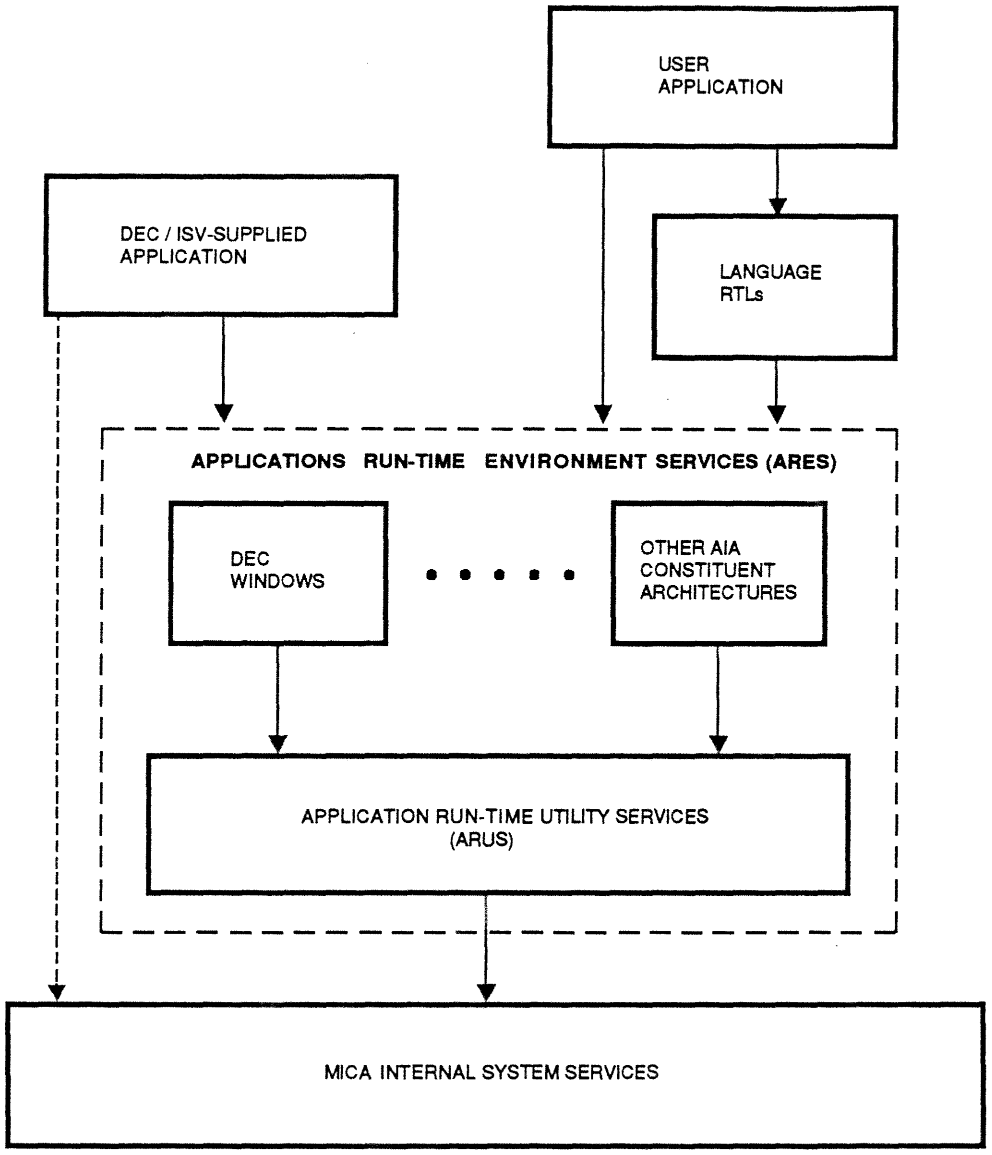
\The desire to not change anything ever is a strong one in certain circles.\

4 What AIA Could Eventually Become on Mica

Before constraining the problem based on annoying realities like schedules and cost, it is useful to describe what the model of AIA on Mica could look like in its final form.

Figure 2 shows a high-level overview of the organization of the full-blown model. Two sample applications are also shown. Please note that this box shows the *logical* layering of these components, not necessarily the actual implementations.

Figure 2: The Full-Blown AIA/Mica Model


USER APPLICATION
DEC / ISV-SUPPLIED APPLICATION
LANGUAGE RTLs
APPLICATIONS RUN-TIME ENVIRONMENT SERVICES (ARES)
DEC WINDOWS
OTHER AIA CONSTITUENT ARCHITECTURES
APPLICATION RUN-TIME UTILITY SERVICES (ARUS)
MICA INTERNAL SYSTEM SERVICES

The top right of the figure shows a typical user application. It makes calls to standard language RTLs as required and it invokes certain visible portions of AIA. The language RTL also implements most, if not all, of its code by invoking core AIA capabilities. There will always be some RTL-specific "magic" that doesn't invoke an AIA component, but in this idealized vision those calls are almost nonexistent.

The top left of the figure shows a typical DEC- or ISV-supplied application. It may also invoke language-specific RTLs, but this figure does not show that. The dotted arrow from the application to the Mica Internal System Services shows that there is a higher probability that DEC- or ISV-supplied code will invoke operating-system-specific features for performance or functionality reasons. There is typically a small performance penalty for using AIA-level routines in most implementations due to the required layering on top of existing facilities. \Mica-CS avoids this in many places by providing the capability directly.\

Providing the foundation at the base of the model are the underlying operating system capabilities, in this case the Mica Internal System Services.

At the core of the model are the individual AIA architectures as described in Roger Heinen's original AIA memo. Many of these architectures are already in development and will be applicable to Mica-CS.

Some AIA architectures such as RPC, Workgroups, DECwindows, and the Compound Document Architecture (CDA) are required for the compute server at FRS. Other AIA architectures such as All-In-1, the Data-Store Management System (DSMS), the Document Database (DDb), and File Cabinet Architecture (FCA) are not required for the compute server at FRS, but will be required when Mica matures to a full-featured operating system.

In either case, this vision description assumes that the current set of AIA architectures will be developed and are eventually applicable to Mica. The focus of this section of this document is on those capabilities that are not currently present in AIA that I believe are important for Mica-CS.

Some of these additional required capabilities are:

- A comprehensive set of portable utility procedures to reduce code dependency on any underlying operating system.
- An AIA-level Calling Standard and other "glue" needed to ensure interoperability of the components when distributed across different operating systems.
- A DECwindows equivalent (or suitable extensions) to support character cell terminals in a distributed manner.
- A portable enhanced file and record access interface.
- The *AIA Programmers Reference Manual*.

The *AIA Programmers Reference Manual* is mentioned here to emphasize the fact that not all of the important deliverables for AIA are software products. This document describes how a programmer can use the various architectures of AIA in order to construct a solution that will be portable across the supported operating systems. It has an overview of the capabilities available and pointers to the detailed operating-system-independent/dependent documentation for each architecture.

Some proposed capabilities are described in the sections below that address the other requirements in the list above. When reading these descriptions, please remember that the following descriptions are a vision unencumbered by real world constraints.

4.1 Applications Run-Time Utility Services (ARUS) †

The logical ancestor of the ARUS is the VAX/VMS Common RTL.

The VAX/VMS RTL is now ten years old. We have been learning about building common environments along the way. We did many things right, but we made a few mistakes that now present us with a library that is tightly bound to the VAX architecture and the VMS operating system. These are mistakes in hindsight: one of the original goals of the VAX software program was to make maximum use of the VAX hardware with little thought of portability.

† None of the names or logical groupings of these services are fixed by any means.

Even without the existence of AIA, pressures have been building for the creation of a portable set of utility capabilities of similar power to the VAX/VMS RTLs:

- We need to provide a set of run time libraries with the power of the VAX/VMS RTL on the Mica operating system and PRISM hardware. Doing this involves a major rewrite of the existing VAX/VMS RTLs; it is not a simple software port.

  For instance, the current VMS RTL lets operating system entities such as channels and PIDs show through the interface, and it also lets the concept of "pages" show through. Things like this make it very difficult to port the user interface without changing the semantics (thereby risking program breakage).
- There is a new corporate direction emerging that Digital is to become a premier provider of ULTRIX software and services. Part of the effort to become such a provider will necessarily involve moving an increasing amount of our VAX/VMS-based software development environment to ULTRIX. This, in turn, requires RTLs.
- A new environment is about to be unleashed on DIGITAL software, through the courtesy of DECWindows and RPCs. In this environment, we need to be smarter about whether a specific operation must be performed in client software or server software. This will involve a change in the interface of some of our routines, particularly those that gather information.

We can best meet these new requirements for portable libraries through the creation of a set of routines that are AIA-conformant and

- provide more general capabilities,
- embody a much higher level of data abstraction in the user interface, and
- provide extended capabilities

over the existing VAX/VMS RTLs. ARUS is this solution. The rest of Section 4.1 describes capabilities present and not present in the full-blown incarnation of ARUS.

In Figure 2 all of the AIA architectures are implemented using the capabilities of ARUS as a base. Not shown in the example, are the possible direct invocation of ARUS capabilities by the two applications and the language RTLs.

4.1.1 Desirable ARUS Capabilities

To create this outline we started with the current VAX/VMS image LIBRTL.EXE, to determine the categories of capabilities it provides. We then added capabilities that are currently missing, but which would make development of AIA-conformant applications and libraries easier.

This document assumes that the reader is familiar with the existing VAX/VMS RTLs. It makes many comparisons between the desired capabilities of ARUS and the existing capabilities of the VAX/VMS RTLs.

4.1.1.1 Virtual Memory Management Routines

One of the most fundamental needs of programs running in a modern system is easy efficient management of heap storage. The LIB$VM family of routines performs this function on VAX/VMS. An AIA memory manager would have the same general capabilities as the current routines, however, there would be no memory management at the page level. Indeed, there would be no mention of memory pages at the user interface level at all: pages are no longer a well defined term. Instead, memory to be allocated would be sized entirely in bytes, and there would be a means to request alignment on an arbitrary power-of-two address boundary.

We believe that such an interface can be portable, simpler, and just as flexible/powerful as the existing set of interfaces.

4.1.1.2 Condition Handling Routines

Allow code to handle a condition in an architecture- and operating-system-independent fashion. These routines do not allow the operating-system-specific manner in which a condition is reported (signal /mechanism vector, etc.) to show through. They should be stack based like the VMS model, not like the ULTRIX model; however, they need to be able to be implemented on ULTRIX.

Required capabilities include:

- Determine the condition and access all condition-specific arguments if any.
- "Pass" on a particular condition, allowing earlier handlers a chance at it.
- Terminate condition handling, resuming normal program execution.
- Modify a condition or one of its arguments.
- Replace a condition and all its arguments.
- Add a sub-ordinate or super-ordinate condition with arguments.

Note that there is no provision for dynamically establishing a condition handler, or removing one from the list. In a cross-OS and cross-architecture environment, this will probably need help from the compilers.

4.1.1.3 Condition Signaling Routines

Provide a way to initiate a condition, similar to LIB$SIGNAL and LIB$STOP. These routines would take an AIA-level condition name, and some condition-specific number of arguments.

4.1.1.4 Process and Thread Manipulation Routines

An architecture- and OS-independent layer for manipulating processes and threads, and otherwise assisting the writers of portable multi-thread applications has been under development in ZK for the better part of a year. It is known as the Common Multi-Thread Architecture (CMA).

This group of routines would also include the capabilities of LIB$SPAWN and LIB$ATTACH, although they are not formally part of the CMA. Exactly how they interact with CMA routines is not defined yet.

\Please see me for a copy of the draft CMA functional specification.\

4.1.1.5 Date and Time Manipulation Routines

Similar to the VAX/VMS V5.0 date/time features, these routines:

- Obtain the current date and time.
- Flexibly format a date or time.
- Convert a textual date/time into the system's internal format.
- Convert a textual date/time into the Universal Time format.
- Perform arithmetic operations on internal format times.
- Perform conversions on internal format times.
- Support international application requirements.

4.1.1.6 String Mapping Routines

The capability of mapping strings similar to that provided by the VAX/VMS logical name services is needed. It should be a hierarchical system providing some level of security if a secure mapping is requested. For low-end systems, it seems that this could be implemented via an RPC server maintaining the name space.

4.1.1.7 String Translation Routines

This category of routines allows the character-by-character translation of strings, rather than the entire string mapping performed by the routines in the previous section. This category of routine would include character set mappings, such as ASCII to EBCDIC.

4.1.1.8 Internationalization Aid Routines

There are currently several aids to assist the writer of international applications, such as:

- LIB$CURRENCY
- LIB$RADIX
- LIB$DIGIT_SEP
- LIB$LP_LINES
- STR$COMPARE_MULTI
- The date/time formatting and parsing routines in VMS V5.0.
- The NCS$ routines for string comparison/sorting.

The same capabilities need to be provided and extended. Possible areas for expansion are:

- Text retrieval - so that applications do not need to embed text in their code. This is currently done by overloading the message mechanism on VAX/VMS.
- Keyboard mapping - Some utilities hard bind functions to a certain keyboard scan code. On non-English keyboards, this scan code may not exist, or may require a compose sequence to generate. \This may actually be a function more appropriate to ARTS described in Section 4.3\
- [More TBS]

4.1.1.9 Data Conversion Routines

We need several broad categories of routines here:

- Atomic-numeric to atomic-numeric
- Numeric-string to/from atomic-numeric and other numeric-string
- Numeric to/from text

These are currently scattered across LIB$, OTS$, MTH$, COB$, and FOR$ (at least). They need to be centralized, standardized, and well documented.

The CDA conversion and access routines (DDIF/DDFF/etc.) are also part of this set.

4.1.1.10 Text String Manipulation

These are routines to manipulate strings, find characters within strings, determine information about strings, etc. The existing STR$ package is a good starting point for this part of the new library, as it is already very portable and well isolated from the operating system and hardware architecture.

These routines should include support for generalized strings required for international support, such as TEXT-16.

4.1.1.11 Common Math Routines

Support for F-FLOAT and G-FLOAT data is standard. Extensions allow for the support of additional floating point types, but these are not supported everywhere.

The equivalents of the majority of the VAX/VMS MTH$ RTL functions are provided for each floating datatype supported. \The voluminous list of candidate functions provided by Jeff Wiener is available upon request.\

4.1.1.12 Command Line Interface Routines

Similar to the CLI$ routines, with extensions as necessary to support ULTRIX-style command lines and integrated with DECwindows by a mechanism TBD. \On VMS, this mechanism is the DECwindows Dialogue Manager. On Ultrix, this mechanism is TBD.\

While the CLI$ routines are not themselves currently part of the utility RTL, there are several utility routines which interface to the CLI such as LIB$GET_FOREIGN.

4.1.1.13 File System Interface Routines

ARUS should provide a basic interface to the underlying file system. The following capabilities should be provided:

- Delete a file.
- Rename a file.
- Create a directory.
- Translate from ARFS-format internal file/record representation into local operating-system-specific forms.

In the full-blown ARUS model, the underlying file and record access system is ARFS as described in Section 4.4.

4.1.1.14 Generic Equivalents of Useful VAX Instructions

There are currently many RTL routines whose purpose is to let high-level language programmers access the full array of VAX machine instructions. As such, they are not portable or suitable for an AIA environment. Many of these routines provide useful features that should be supplied in a manner not tightly coupled to VAX. For instance, extended arithmetic, CRC calculation, contiguous memory moves, queue manipulation, etc. should be provided.

4.1.1.15 Multiprecision Arithmetic Routines

Routines to perform multi-unit integer arithmetic. Similar to LIB$ADDX and friends.

4.1.1.16 Tree Manipulation Routines

These should be similar to the LIB$ binary tree routines with some extensions, most notably LIB$REMOVE_FROM_TREE.

4.1.1.17 Graph Manipulation Routines

Tools for manipulating arbitrary graphs (DAGs). Possibly evolved from the GRAPHER package, or from the net-manager work planned to be done in SDT as part of the DECWindows layers.

There are currently no graph manipulation routines in the Utility RTL.

4.1.1.18 Cross-Reference Routines

The current CRF$ routines are essentially unused, and every compiler writes its own cross reference routines. There is obviously something wrong with the current package, and obviously a need for a centralized, supported package that fills the requirements of the various compilers and other utilities. More research is needed.

4.1.1.19 Table-Driven Parsing Routines

Tools such as LIB$TPARSE to write simple parsers.

4.1.1.20 Run Statistics Routines

This group is frequently referred to as "Performance Measurement", but that is a bit overstating the capabilities of the current routines. These routines return items such as the virtual memory get/free statistics, or elapsed CPU time since the last call.

\How useful these routines are, and what is possible in the cross-os and cross-architecture environment are open questions.\

4.1.1.21 Process/Thread Information Routines

These routines return information to the program about its environment and its past execution. Some of the types of information returned include:

- CPU time consumed
- AST level (or some equivalent- as "AST" may not be a portable concept)
- [More TBS]

4.1.1.22 Resource Management/Synchronization Routines

There is a need for some form of resource management routines, such as the LIB$xxx_EF routines. The CMA specifies capabilities to declare, initialize, and use semaphores and other types of synchronization objects. That may be the extent of support, since most other resource managers can be built from them.

4.1.1.23 Image Management Routines

Routines in this group provide a way to activate new images, either merging with or replacing the currently executing image.

4.1.1.24 Operator Communication Routines

Some simple, portable capabilities are required here to standardize the program interface with any operating-system-specific operator communications facility.

Required capabilities include:

- Sending a message to the operator.
- Sending a message to the operator and waiting for a response.
- Canceling a previously sent request.

4.1.2 Capabilities Excluded from ARUS

There are several categories of routines that exist in the current VAX/VMS RTLs that should NOT be made a part of ARUS.

4.1.2.1 Obsolete Routines

There are several routines still shipped and supported that are, for one reason or another, obsolete or undocumented. Clearly these routines should not be provided; they should be allowed to die.

4.1.2.2 JSB Entry Points

Many routines have a JSB entry point as well as a CALLx entry point. This is a non-transportable optimization that is counterproductive in a library whose main goal is to be portable.

4.1.2.3 Other Routines With Multiple Entry Points

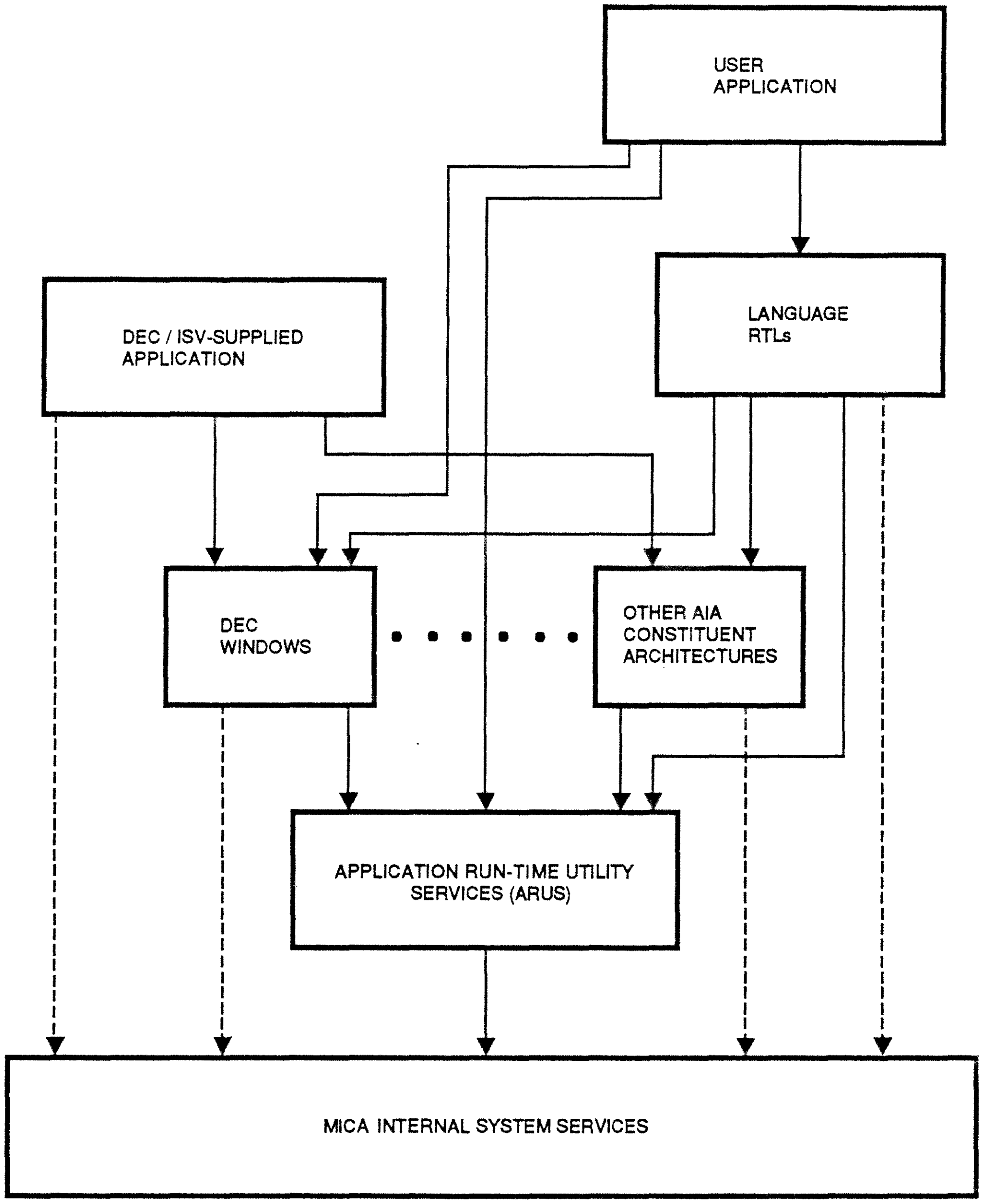
Other routines have many entry point names for identical or highly similar capabilities. The cases of *identical* capabilities have typically evolved from moving language-specific routines into the language-independent libraries. The cases of *highly similar* routines have typically evolved due to the fact that the LIB$, STR$ and OTS$ facilities have differing condition reporting semantics.

It is a goal that the new library have exactly one routine to accomplish a particular action.

4.2 Applications Run-Time Environment Services (ARES)

In Figure 2 a certain level of detail is hidden inside the dotted line box that represents the Applications Run-Time Environment Services (ARES). In Figure 3 more of the underlying interconnections are shown to help explain what ARES is.

Figure 3: AIA Without ARES


USER APPLICATION
LANGUAGE RTLs
DEC / ISV-SUPPLIED APPLICATION
DEC WINDOWS
OTHER AIA CONSTITUENT ARCHITECTURES
APPLICATION RUN-TIME UTILITY SERVICES (ARUS)
MICA INTERNAL SYSTEM SERVICES

Notice that Figure 3 shows the same basic components as presented in Figure 2. The basic relationships of the applications, language RTLs, AIA core architectures, and underlying operating system support are the same as well.

ARES is the "glue" that gives the full-blown model shown in Figure 2 its seamless nature. ARES is as much a statement of the completeness and maturity of ARUS and AIA capabilities as it is the presence of any software product. As such, it is hard to point at a particular piece of software and say "that is ARES."

Figure 3 can be considered to be simply a picture of the typical interactions of the components in a less-developed model of AIA and ARUS. That is why the box representing ARUS in the Figure 3 is smaller than in Figure 2; it is less capable.

The many solid arrows emanating from the two example applications are meant to show that the interfaces to various AIA components appear to be dissimilar to the applications programmer. The additional dashed arrows in the picture are meant to represent the reality of more software that directly invokes the capabilities of the underlying operating system. For example, the dashed arrow running from the language RTLs to the Mica Internal System Services box might represent the fact that the RTLs actually use Mica's process and thread manipulation capabilities due to a limitation in ARUS.

At the level of the core AIA components, this lack of seamlessness means that two equivalent AIA-level components don't have the same style of interface. For example, because of its UNIX† and C heritage, DECwindows reports exceptional events back to its caller in a different manner than other AIA components do. Eventually, all AIA components will use a common mechanism to report exceptional events to their callers.

In the future, ARES gives a definition of what *approaches* a portable operating system‡ in order to make all of the interfaces to AIA consistent. \This is clearly a long-term goal, when AIA is *very* mature.\

Logically, ARES contains the following:

- An AIA-level Calling Standard.
- AIA-level abstractions of process and thread.
- An AIA-level condition handling mechanism.
- An AIA-level capability for application flow control.

The point of this discussion and Figure 3 is that until that maturity is reached, there will be a level of clutter to AIA.

Some of the items in the requirements list above are not well understood today and do not have to be present at the compute server FRS. Because of that, my thoughts on this are definitely sketchy.

4.2.1 An AIA-level Calling Standard

Initially this may be nothing more than the VAX Calling Standard coupled with constraints appropriate to early RPC implementations. That will get us a long way.

As ARES matures, however, some basic issues related to the interoperability of procedure interfaces across many diverse architectures need to be addressed. Capabilities in the AIA Calling Standard to address these issues might include definitions of:

- Common supported datatypes.
- Simple architected argument coercions allowed for compatibility.

  For example, an AIA-level procedure definition defined with the AIA Calling Standard might specify that an integer input argument to the procedure is expressed as a common "natural" integer size. Transparent coercion of such arguments to the natural size of integer appropriate to the implementation of the procedure body would be covered by the calling standard.

† UNIX is a trademark of AT&T

‡ There, I said it.

- The AIA-level condition handling briefly described in ARUS is fleshed out an implemented.
- Much more TBD.

Although not as much a technical concern, the calling standard should also specify the "look and feel" of AIA-conformant interfaces. This is to promote the consistency of interface that contributes to the perception of AIA as a single (if incredibly large) facility.

4.2.2 AIA-level Application Flow Control

With the existence of new user interfaces such as DECwindows, what capabilities are needed to control the scripted execution of several distinct programs? Command files and shell scripts aren't sufficient.

Is providing for callable entry points for all DEC-supplied utilities sufficient for allowing for flow control via program generation? I think not. There will always be the need for a conceptually simpler mechanism that doesn't require programming skills.

4.3 Application Run-Time Terminal Services (ARTS)

Character cell terminals, including hardcopy terminals, are not going to disappear from the scene in our lifetime.

ARTS contains capabilities designed to allow for the portable support of character cell terminals, including foreign terminals, in a distributed fashion.

Note that ARTS is not envisioned to be a forms package; it is possible that ARTS could be used as a basis for support of a forms package, but absolute forms performance requirements have typically prevented this type of logical layering in the past.

4.3.1 Basic ARTS Capabilities

In its simplest form ARTS allows for input and output operations that are suitable for use on hardcopy terminals or softcopy terminals with one "window" that is the entire screen. That is, it is possible to perform a read or write operation without creating a supporting virtual display/pasteboard/etc. for this simple case.

Note that ARTS does not provide for support of hardcopy terminals in other than this simple sequential, implied position model. There is no support for transparent emulation of softcopy terminal fallback presentations.

ARTS does add value to this simple model by providing a mechanism to invoke hardware-specific marking capabilities in a portable fashion, that is, without requiring an application to embed device-specific control sequences in output strings.

Input key mapping and function key definition support is also included.

4.3.2 Enhanced ARTS Capabilities

These capabilities include the ability to create multiple logical "windows," decorate those windows with output marking instructions, manage the input focus of a keyboard associated with those windows, and manage the terminal real-estate while presenting the logical "windows" on the physical display screen. The capabilities correspond roughly to the SMG$ routines provided by VAX/VMS today.

It would be desirable to tie the ARTS capabilities to utilize the DECwindows distribution capabilities. It may be possible to specify ARTS as a unique implementation of a limited set of DECwindows calls plus extensions to DECwindows using the DECwindows extension architecture. Much more research is needed.

4.4 Applications Run-Time File Services (ARFS)

ARFS is designed to provide a portable distributed file and record access program interface with enhanced capabilities. ARFS raises the base file and record access support provided in the AIA target operating systems to a level of capability roughly equivalent to RMS today.

Unlike RMS, however, ARFS has an AIA-conformant interface and has additional capabilities to provide an end-user interface that includes the Data-Store Management System (DSMS) capabilities in a consistent interface.

These DSMS capabilities allow for the definition of simple hierarchical file organization schemes with user-extensible file attributes. This level of support obviates the requirement for programs with simple databases, such as MAIL or NOTES, to re-implement these extensions to extend the base underlying file and record access support. DSMS capabilities are also eventually required to support the File Cabinet Architecture (FCA) which in turn supports the eventual execution of All-In-1 on an interactive Mica system.

ARFS provides the glue between DECwindows, ARTS, and the file system interface for simple terminal- or window-related I/O. That is, ARFS can direct simple sequential terminal output and input to/from DECwindows and ARTS-interfaced terminals.

ARFS also defines a mechanism for directing output to the default input output, and error streams in an operating-system-independent way.

\How is this different from the Distributed File Service (DFS) component of workgroups? Is this just an AIA-conformant interface to DFS?\

5 Proposed Steps in the Implementation of AIA on Mica

Based on the assumptions listed above about the customer requirements at FRS, the following three phase implementation plan is suggested.

Three arbitrary release of Mica-CS are considered:

- Mica-CS V1 ("Nuts-and-Bolts capabilities for FRS.")
- Mica-CS V2-V? Functional Releases ("Filling in the gaps.")
- Mica-CS V*interactive* ("Full-featured interactive Mica system.")

For the first release, estimates of the relative complexity of the work and how the work could be accomplished are provided.

It is important to note that new AIA capabilities designed for Mica-CS will need to be reviewed by representatives of the other target operating systems; this will add time to the design review process.

It is also important to note that in order for the Mica-CS implementation to be perceived as seamless, these capabilities need to be provided on ULTRIX and VMS at the same time. This document should not be construed to imply the existence of formal or informal commitments on the part of any other Digital development group to provide any such capabilities.

5.1 Mica-CS V1 Release

I assume that the following components discussed in other Mica-CS plans will be present at FRS:

- RPC
- Workgroup distributed security capability
- Workgroup distributed naming capability
- Workgroup distributed file service capability
- Language-specific RTLs for FRS languages

Table 1 lists the AIA components and sub-components that should be provided for Mica-CS V1. This list is presented in general descending order of importance by component and subcomponent.

The column labelled "Complexity" shows two estimates of the work involved. The first value provided is the level of complexity of the definition/design work expressed on a scale of "High-Medium-Low". The second value provided is the amount of work involved to create/port the software; a scale of "Large-Medium-Small" is used.

Table 1: AIA Candidates For Mica-CS V1

Component	Subcomponent	Work Required	Complexity	Resources
ARUS	Process/Thread Manipulation and Information Routines, Resource Management /Synchronization Routines, Condition Handling and Signaling Routines	Architect and implement using CMA and previous Mica work as a starting point. Define process and thread level abstractions.	High/Medium	DECwest, SDT, AIA Design Reviewers
	Virtual Memory Management Routines	Architect and implement.	Medium/Small	DECwest, SDT, AIA Design Reviewers
	Common Math Routines	Much more basic work about the possibility of a portable math library needs to be done. Architect portable math environment and implement.	High/Large	SDT, DECwest, AIA Design Reviewers
	File System Interface Routines	Make decision about ARFS versus DFS. Architect and implement.	Medium/¿Medium?	DECwest, AIA Design Reviewers
	Text String Manipulation	Architect and implement.	Low/Medium	SDT, DECwest, AIA Design Reviewers
	Data Conversion Routines	Architect and implement.	Low/Small	SDT, DECwest, AIA Design Reviewers
	String Mapping Routines	Architect and implement.	Medium/Small	SDT, DECwest, AIA Design Reviewers
	String Translation Routines	Architect and implement.	Low/Small	SDT, DECwest, AIA Design Reviewers
	Date and Time Manipulation Routines	Architect and implement.	Medium/Small	SDT, DECwest, AIA Design Reviewers
	Command Line Interface Routines	Design and implement using CLI$ routines and previous Mica work as a starting point.	Medium/Medium	DECwest, SDT, AIA Design Reviewers
DECwindows	XLIB Xtoolkit DECtoolkit	Research complexity of porting. Design and implement new DECnet-MICA transport code. Port 400-500 C functions.	Low/Big	DECwest
DECwindows (ULTRIX)	Ensure basic interoperability with ULTRIX	Research TCP-IP transport interface. Design and implement ULTRIX "pseudo-server."	Low/¿Small?	DECwest
	Ensure interoperability with ULTRIX User Executive	ULTRIX User Executive specification unavailable. Review it when it becomes available.	Low/¿Small?	DECwest
ARTS		Research feasibility of using DECwindows as basis for ARTS. Define ARTS using SMG$ as a starting point. Implement basic ARTS capabilities.	High/Medium	DECwest, SDT, AIA Design Reviewers
CDA	DDIF Conversion RTL routines	Research port complexity.	Low/Low	DECwest, BOSE
Print System Model	Submitter Subroutine Interface	Interface is currently undefined. Participate in definition. Provide RPC stubs for execution of Submitter Subroutines on client	Low/Small	PSM Task Group, DECwest
ARES	Calling Standard	Research and begin development of Calling Standard	Medium/¿Large?	DECwest, SDT, AIA Design Reviewers
ARFS		Determine exact nature of relationship of ARFS with DFS and RMS. Is there a real need for ARFS?	High/¿Large?	DECwest

5.2 Mica-CS V2-V? Releases

The following AIA components and sub-components should be provided in these releases, to enhance the AIA environment or provide the framework for V*interactive*:

- Complete ARUS definition and implementation of the following classes of routines:
  - Cross-Reference Routines
  - Generic Equivalents of Useful VAX Instructions
  - Graph Manipulation Routines
  - Image Management Routines
  - Internationalization Aid Routines
  - Multiprecision Arithmetic Routines
  - Operator Communication Routines
  - Run Statistics Routines
  - Table-Driven Parsing Routines
  - Tree Manipulation Routines
- Implement mechanism for DECwindows architected extensions.
- Implement GKS and PHIGS DECwindows extensions.
- Complete implementation of ARTS capabilities not present in V1.
- Implement DSMS in preparation for V*interactive.*

5.3 Mica-CS V*interactive* Release

The following AIA components and sub-components should be provided at this release:

- DECwindows XSERVER, User Executive, and assorted interactive utilities and as-yet-undefined extensions. For example, provide the desktop tools.
- Compound Document Architecture: Assorted interactive EPIC utilities. For example, provide the Compound Document Editor.
- Assorted Print System Model components. For example, host the execution of the Print Symbiont.
- Assorted Integrated Programming Support Environment (IPSE) components.
- All-In-1 and supporting File Cabinet Architecture and Document Database components.
- Continuing implementation and retrofit of ARUS improvements related to ARES maturity. For example, retrofit DECwindows with a true AIA-conformant interface.
- Provide ARES flow control capabilities.

Most of these components have not been provided before this release because they apply only to interactive capabilities. Some have not been provided because they are in very early stages of definition at the present time and only solidified after the Mica-CS V1 timeframe.

6 Some Closing Observations

- All of ARES doesn't have to be in place at the FRS of the compute server.

  We will choose to place emphasis on providing certain components such as DECwindows and portions of ARUS at FRS. Other components can be added later.

- Even though all of ARES won't be in place for several years, we can anticipate it and build the foundations for its eventual maturation.

  We don't have to solve every problem before we start, however. One purpose for my writing this document has been to share the fact that a lot of definition work has to be performed on these new capabilities (AR*S). I believe that we will be missing a large opportunity if we allow concerns about the volume of that work to deter us from an aggressive program of involvement in AIA.

- There will always be a need for supporting certain Mica- or PRISM-specific capabilities in user-visible (maybe ISV only-) interfaces that will not be part of AIA.

  Capabilities that are not present in ARUS at FRS will have to be provided on a temporary (no such word) basis by Mica.

  It is a goal for Mica-CS to minimize these cases, however. It is unreasonable to expect that we will be able to avoid some level of documentation regarding these capabilities.

- No matter how fast we deliver AIA components, there will be a potentially large lead time before customers acknowledge that AIA is valuable.

  That lead time should be anticipated and used to flesh out the AIA offerings across all of the supported operating systems. This time will also give us a chance to retrofit certain pre-existing architectures to match the fully developed model of AIA that we'd like to ultimately achieve.

- Reducing that lead time and enhancing the acceptance of AIA is as important as delivering the AIA components themselves.

  The message about Digital's commitment to AIA has to be clear and supported by actions demonstrating that commitment. ISV's particularly need to be qualified and encouraged to participate in seed programs, technical exchanges, etc. Good documentation targeted to the needs of ISV's is a must.

- AIA capabilities cannot afford to be "least-common-denominator" capabilities.

7 Acknowledgements

This document was the result of a collaboration between the following individuals, based on a minimal amount of direction from the author:

- Al Simons - ARUS capability descriptions
- Jeff Wiener - Math RTL input
- Laurie Dawson and Craig Kosak - Artwork

This document does not reflect their views on all issues.

My thanks to all who helped or listened to the nth recitation of the oral version of these thoughts.

The Modula-2+ synchronisation primities are described in SRC report Nr 20, available online in Circus::SRC$notes:t6.ps, and also in the 11th SOSP proceedings

A condensed version of the CMA spec is available in the CLT::Threads notes file, note 31.5, or in HOBART::CONDESNSED.SPEC (sic).

```
From:   DECWET::COCKCROFT "Claire Cockcroft, DECwest Engineering  03-Feb-1988 09
To:     BECALM::NYLANDER,COCKCROFT
Subj:   Environment Information Requirements for Glacier
```

DIGITAL INTEROFFICE MEMORANDUM

TO:	distribution	DATE:	February 3, 1988
		FROM:	Claire Cockcroft, Dennis Doherty
		DEPT:	DECwest Engineering
		EXT:	206-865-8916
		LOC:	ZSO
		ENET:	DECWET::COCKCROFT

cc: Mark Ozur
Benn Schreiber

Digital Equipment Corporation--Confidential and Proprietary

SUBJECT: Environment Information Requirements for Glacier

The Glacier product is a compute server that provides access to Mica system resources through an integrated client /server interface. The client/server mechanism for executing applications in the compute server environment is divided into a client portion of support software (the client context server) and a Mica portion (the job controller server). The client context server and the job controller server communicate via remote procedure calls. It is the responsibility of the client context server and the job controller server to set up the environment in which a user's application will execute.

The following questions arise about the execution environment:

- o Exactly what items of information from the client system should be included in the execution environment?
- o Where should items of environment information be located for access by the user's application?

We have already identified some environment requirements. For instance, Mica RMS requires the user's default volume and default directory. In the current design, the client context server calls the job controller server to create a Mica job. Before creating the job and thus causing the user's image to begin execution, the job controller server requests Mica RMS information from the client context server, and stores the RMS information as Mica logical names in a process container for use by Mica RMS. Likewise,the status/message/text-formatting facility requires the user's default natural language and default status message format. These two items will be stored as Mica logical names using the same mechanism as Mica RMS items.

It is not clear exactly what environment information is required by the language runtime libraries. C, for instance, expects access to command line arguments and environment variables. There are several ways in which this information could be supplied; the following describes three possible mechanisms:

1. The client context server may pass the information to the job controller server when it requests a job creation. The job controller server may then store the information in the Mica process_data_block parameter when it calls exec$create_job. The compute server support software does not attempt to differentiate one language from another, and thus stores the same information for all user programs.

2. Following the mechanism outlined above for Mica RMS information, the client context server may fetch information upon request from the job controller server, which stores the information as logical names. Again, the compute server support software stores the same information for all user programs.

3. Runtime libraries, specific to the programming language, may request environment information through remote procedure calls to the client context server during initialization routines (prior to user code execution). The client context server would have to provide support for these language specific procedures.

In order to decide which, if any, of the above solutions is the best design for Glacier, it is important that we identify the specific environment information requirements of the language runtime libraries. And to do this, we need your input.

Please review your language runtime library requirements for the Glacier compute server model. In your consideration include requirements for both FRS and later releases, keeping in mind such parts of the execution environment as logical names and context services. If you have specific expectations, presumptions, or requirements, please send them to me no later than February 15th, for inclusion in our design.

Dennis Doherty and I are available to answer questions about the current design of compute server support software. If you know of anyone who may have additional requirements, please forward this memo to the appropriate party.

DISTRIBUTION:

Dave Ballenger
Chip Nylander
Darryl Havens

follow up

```
From:   TLE::VNX::KEATING      29-JAN-1988 14:28
To:     CHIP
Subj:   This should be embedded in Prism Files at FCS.  Will you sponsor? /Bill

From:   PIXEL::TRAVIS "Bob Travis, ZKO2-1/N20, 381-2762  29-Jan-1988 1104" 29-JA
To:     @CDPAC-MEM,@CDPAC-INT
Subj:   first-level info on VMS plans to support DDIF

To: CDP-AC members
cc: interest

Here is early information on planned level of VMS support (in 5.2) for
DDIF files via RMS extensions.  If you have any questions, please send
them to Stu.

Thanks,
Bob

----------------------

From:   STAR::DAVIDSON     "Stu Davidson" 29-JAN-1988 08:51
To:     @TAG.DIS
CC:     DAVIDSON
Subj:   Minutes -- VMS support for DDIF files




        +---+---+---+---+---+---+---+
        | d   i   g   i   t   a   l |   I N T E R O F F I C E   M E M O
        +---+---+---+---+---+---+---+


        TO: distribution                    MEMO: VMS support for DDIF files
                                            DATE: 29-JAN-1988
                                            FROM: Stu Davidson
                                                  Trevor Kempsell
                                            DEPT: VMS
                                            LOC:  ZK03-4/Y02
                                            NODE: STAR::DAVIDSON, STAR::KEMPSELL

Distribution:
        REM::ARANDA
        DSSDEV::BUTLER
        DSSDEV::CHASEN
        STAR::DAVIDSON
        STAR::GEORGE
        DSSDEV::HALLGRIMSSON
        DSSDEV::JACK
        STAR::KEMPSELL
        STAR::KENAH
        STAR::NIGEL
        STAR::PENNINGTON
        STAR::SCHAEFER
        STAR::STEEVES
        PIXEL::TRAVIS

        SUBJ: Meeting between VMS and
                Compound Document Program interest group
```

13-Jan-1988

Support in VMS

VMS support for DECwindows, including any support for DDIF encoded files, will not be included in VMS V5.0.

Any changes in base VMS components, required for DECwindows V1.0, will be shipped as part of the DECwindows V1.0 kit, which will not be a VMS release.

Base VMS components which ship with DECwindows V1.0 will be merged into VMS with the v5.2 release. Base system support for DECwindows in 5.2 will be at least equal to the support provided in DECwindows V1.0.

Tagging

In keeping with the current VMS policy of not depending on file naming conventions, it was agreed that some method of absolute identification of a file as DDIF (from the file header, not data inspection) is required.

The scheme for 'tagging' files recommended by the Compound Document Architecture board is use of the ASN.1 object identifier. A typical length of an object identifier is on the order of 12 bytes.

It was further agreed that, within VMS, all DDIF files are to be tagged, and applications will not treat untagged files as being encoded in DDIF. A DDIF file which is not tagged will be considered corrupt, and require some type of repair.

SIGNIFICANCE OF DDIF

(I stole this section from Ron Schaefer. Even though these words may not have been said at the meeting, I think this captures the underlying level of concern.)

The only significant question is: "How important will DDIF be?" or more precisely: "How popular will DDIF files be?".

Unfortunately no one really knows, but I think some consideration of the consequences is worth discussing:

Suppose DDIF never becomes very popular and remains as a file format for a small, fringe set of window programs.
    An implementation that is based on some kind of escape and off-to-the-side support is commensurate with the need.
    An implementation that properly int[illegible] DDIF into VMS & RMS is overkill work but still useful.

If, on the other hand, DDIF becomes a mainstream file format and occurs frequently then:
    An implementation that is based on some kind of escape and off-to-the-side support will be a disaster with respect to making

VMS & DDIF look coherent.
An implementation that properly integrates DDIF into VMS & RMS is necessary in order to have a decent product.

There is no problem with having the initial implementation (due to time-pressure) supporting less than the full design and no real problem with using some escapes, etc. PROVIDED the basic integrated design is sound and fully thought thru.

It was agreed that, given the significance of DDIF, direct access to DDIF files by existing applications must be supported.

Existing applications

Since no applications which use DDIF format files have yet been shipped, it was agreed that existing applications (VMS components, layered products, third party, or customer written), which could reasonably deal with data in DDIF files fall into one of two categories:

1. Applications which expect to read a sequence of ASCII records. DDIF files can contain data of this sort, and this is the file type which will be produced by the normal editor elected to be used in the DECwindows environment. DDIF files, however, are encoded quite differently from traditional sequential text files within VMS. For existing applications to process text records directly from DDIF files, some filter, or translator, must be employed.

2. Applications which deal with any file type, simply as a sequence of bytes. These are generally file utilities, such as copy, compression, or encryption utilities.

Any applications capable of dealing with DDIF files in native mode fall into the category of 'new applications'.

The central issue in VMS support for DDIF files is: what should happen when an existing application opens a DDIF file? Type 1 applications will fail in unpredictable ways, unless a transparent 'filter' is available. The alternative is that the user must be conscious of file formats.

If a filter is applied for type 2 applications, they may fail to behave as expected.

It is clear that no solution will be correct always. It seems reasonable to adopt an approach which will frequently 'work', is predictable, and provides some work-around for problems.

New Applications

Here, new rules can apply. New applications [illegible] sonably be expected to understand the new rules.

A new application which may wish to deal with new file formats in other than 'default' mode, must specify that intent when opening files.

Agreed approach
----

1. Files will be tagged, through the RMS interface. Normal VMS utilities will preserve the entire contents and tag for DDIF files.

2. Existing applications opening tagged files:

in record mode:
  will get variable length ASCII records, through (an extended) RMS.

in block mode:
  will get unfiltered disk block images.

This should allow most existing applications to work predictably. Tagged files cannot be read in record mode unless an appropriate extension is available (i.e., the $OPEN will fail).

3. New applications opening tagged files:

"New" applications will be identified on $OPEN, when they request the file tag. These applications, at $CONNECT, may specify the semantics desired for reading the file. When the stored file semantics and the desired semantics match, unprocessed records from the file will be passed to the application (no RMS extension used). If the stored and desired semantics differ, RMS will attempt to locate a translator (extension). If no translator is found, the $CONNECT will fail.

4. Printing

It was agreed that printing is not a major issue.

Existing print symbionts will read ASCII records through RMS and existing 'type 1' applications. The Printing System Model will provide a 'new' style application, which will recognize and deal with tagged files. DECwindows will also provide a DDIF to POSTSCRIPT translator.

5. Network file copy

It was agreed that DAP must be extended to allow the exchange of file tags. VMS RMS/FAL will support the new DAP messages. The strategy requires MS-DOS and ULTRIX FAL's to also cooperate.

Rem Aranda accepted responsibility for encouraging [illegible] cooperation.

Further details on the VMS implementation will be forthcoming.

This will include:

  o A list of utilities which will include support for DDIF in
    the DECwindows V1.0 release, to complement the support which

will be shipped in RMS with the V1.0 release.

- VMS will also define its behavior with systems in a network which do not support file tagging.

- Changes to VMSMAIL for DDIF support.

```
From:   TLE::WHITLOCK     "Stan Whitlock  DTN: 381-2011" 15-FEB-1988 12:54
To:     CHIP,RICH,JB,DAVEM,CLT::SIMONS,CLT::GREENWOOD,MATT,CLT::WIENER,WOOLY,WAL
Subj:   minutes of our meeting on Environment Information Requirements for Glaci
```

On 10-Feb-88, the following people met to discuss Environment Information Requirements for Glacier:

John Bishop	Al Simons
Walt Carrell	Jeff Wiener
Dave Moore	Stan Whitlock
Gerald Sacks	

This meeting was prompted by a memo from Claire Cockcroft and Dennis Doherty of DECwest, requesting our input on environment information needed in the client/server model.

We made the following points:

o In general, the program running on the server may want environment information (eg, a logical name translation) from the client at any time during its execution so there must be facilities to both query and set the client environment from the server at any time during execution.

  It may be possible to pass some environment information from the client to the server when the server process is started (as a performance enhancement) but this "improvement" should be transparent to the server process' ability to query/set the client environment.

o We are assuming that the message file resides on the server and that formatted text strings (not status codes) are passed from the server to the client.

  It wwas pointed out that the user will want to be able to get enough status information about a failure to understand where the failure occurred, ie, failure on the server or on the client. This runs opposite to the "seamless" environment which does not require the user to know where his job is running.

o The categories of environment information that we felt must be available included:

  - RMS file name translation
  - RMS settable parameters
  - command line info
  - logical names
  - message state: format, natural language
  - security profile: ACLs and rights identifiers
  - appropriate quotas - we're not sure which ones
  - process information - GETJPI

Claire was in ZK last week and met with me after this group met. She shed some light on the questions of quotas and JPI info between the server and the client. The current thinking is that a very smart system manager (human being) on Glacier will create the proxy account that the client will run under so that the proxy has all of the correct rights and quotas. These may be and probably will be different from the quotas and rights on the client. The coordination of these proxy accounts will be manual and therefore, potentially error-prone.

Claire didn't hear anything radiaclly new and bizarre from our deliberations -

her group had thought of most of this stuff. Al will be out at DECwest this week for the program review and will (no doubt) have a chance to elaborate on this topic.

/Stan