OpenVMS Technical Journal

V10, June 2007

Table of Contents

Strategies for migrating from Alpha and VAX systems to HP Integrity server systems on OpenVMS

Robert Gezelter, CDP, CSA, CSE, Software Consultant

Overview

The ideal method for migration to OpenVMS on the Intel® Itanium® architecture is to recompile, relink, and requalify. For many users, this strategy has been highly successful. Many applications, each comprised of hundreds of thousands of lines of source code, have been ported virtually overnight.

Other organizations face more challenging circumstances. Restricted budgets, operational commitments, dependencies on layered products, staff shortages, the sheer number of applications, and the size of the aggregate source bases make the "all at once" strategy infeasible.

OpenVMS provides unique capabilities that allow an organization to assimilate HP Integrity servers in a phased, structured, extremely low-risk approach. This strategy decouples the different phases to the maximum extent possible, permitting the assimilation process to proceed transparently to users, with minimal disruption to operations and minimal business risk.

OpenVMS has clustering and image translation capabilities that offer end-user management and technical staff a unique degree of flexibility. In the past, this flexibility has enabled OpenVMS to confound critics by enabling the assimilation of Alpha processors and, more recently, Intel Itanium processors. The same capabilities that enabled the rapid assimilation of radically different hardware architectures by the OpenVMS Engineering organization itself are fully available to independent software vendors and end user

Introduction

Corporate Information Technology (IT) is a complex choreography of many interacting assets, including people, hardware, packaged software, and locally developed or maintained software. In today's environment, where IT has become the backbone of the enterprise, corporate IT is, in effect, committed to providing access to a wide variety of systems on a continuous basis, 24 hours a day, 7 days a week, 365 days a year, without interruption.

Changing hardware platforms is one of the most anxiety-provoking transitions in IT. The mere mention of a change of hardware platform brings forth images of long, uncertain migration plans and large numbers of modifications. The resources required for such projects are substantial, and the resulting budgets are commensurate.

While these reasons for concern are reasonable in a general sense, they do not apply to OpenVMS. This article examines the issues involved in the migration of applications on OpenVMS from computing systems based on the reduced instruction set (RISC) Alpha architecture to HP Integrity servers based on the Intel Itanium architecture. OpenVMS provides a set of unique facilities to make the transition from the Alpha architecture to the Itanium architecture a controlled and predictable evolution, with low technical risks. As a result, the transition can be managed with a minimum of business risk. OpenVMS on VAX, Alpha, and HP Integrity systems offers several facilities that allow the management of platform change on an incremental basis. When used properly, these facilities reduce transitions to a technical exercise with controlled steps. They eliminate the largest risk in IT management: the cutover. Instead of choreographing a cutover that affects all users simultaneously, the problem is reduced to transitioning users from the old to the new in gradual incremental groups, commensurate with the available support.

The nature of transition risk

Platform changes have been one of the classic nightmares of IT, [1] resulting in missed commitments, expensive overtime, interruptions of availability, and other dramatic side effects. It is no wonder that IT executives look upon such changes with all the enthusiasm of facing Dracula in person.

Different risks for different organizations

First, it is necessary to fully appreciate the process of platform transition in an end-user environment. Transitions in end-user environments are fundamentally different from transitions in product development environments, such as an independent software vendor (ISV) or original equipment manufacturer (OEM). In an ISV or OEM environment, the workload is skewed toward technical and product issues. Quality control and testing are important, but they are qualitatively different from their counterparts in an end-user organization. By contrast, an end-user organization has more pressing commitments to operations, as well as a lower ratio of engineering resources to programs.

Software development organizations often have extensive source control and rebuild processes, as well they must. Rebuilding products en masse is simply a fact of life. By contrast, end-user organizations are, even in these days of Sarbanes-Oxley, [2] far less likely to have the controls and processes in place to rebuild all used programs from their respective source components. This fact was encountered repeatedly during the remediation of Year 2000-related problems.

In reality, all computing facilities combine aspects of development and production (Figure 1). From a high-level perspective, all facilities can be categorized as an amalgam of the two extreme cases. In reality, it is not unusual to find that the exact categorization of an application, or a component of an application, depends on the role played by the particular component. For example, a compiler is a production component in an organization that is devoted to development, and is just as critical as a production application in an installation that does not undertake software development.

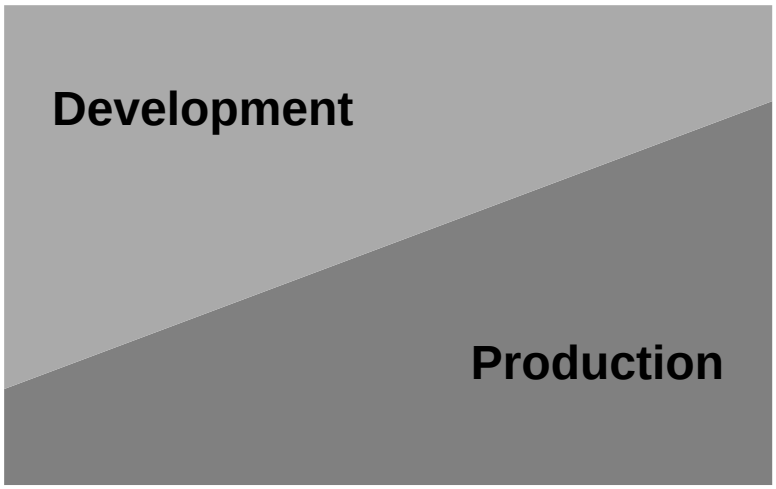


Figure 1 – All computing facilities are some combination of development and production.

As noted, platform replacement is as much a management exercise as a technical exercise. In end-user environments, tasks are weighted toward management and qualification issues. In companies with hundreds or thousands of users, even a minor problem during a cutover can be

[1] The worst nightmare is generally considered to be a total "loss of data" event, such as that experienced by many small and medium businesses in the aftermath of some disaster, such as the World Trade Center attack (September 2001) or Hurricane Katrina in New Orleans (September 2005).

[2] In the United States of America, the Public Company Accounting Reform and Investor Protection Act of 2002 (Public Law 107-204), generally referred to as Sarbanes-Oxley, after its authors, provides the legal basis by which the Chief Executive Officer of a publicly owned corporation must personally certify the accuracy of all financial results on a regular basis.

catastrophic. In concept, the issue is simple: user support is a finite resource. Ideally, one should not cut over more users than can be supported in the event of a problem. The challenge is how to realize this management imperative within the realm of technical feasibility.

The strategic approach advocated by this article is, in turn, useful to OEMs and ISVs, in the context of shortened time and cost to market of products for use on HP Integrity servers running OpenVMS. The same concepts that apply to end users permit ISVs and OEMs to achieve delivery schedules with a low cost, low risk, and high certainty.

Management is rightly concerned with equipment costs and availability, additional problems that are often associated with the acquisition of a new hardware platform and the retirement of older hardware. Once schedule commitments are made, changes are often difficult and expensive. Older equipment might be nearing end of lease, and lease extensions can be expensive. A small slip in migration schedule can be disproportionately expensive. There is often a financial flashpoint between technical and cost considerations, with the technical team eager to defer migration to reduce technical risk, and the management team eager to retire older equipment to reduce financial exposure.

The general concept that operations must be sustained during a transition is far older than general-purpose computing. [3] Business crossed the line into continuous availability more than a century ago, with the advent of the telephone. In 1930, the 12,000-ton Indiana Bell Telephone Building in Indianapolis, Indiana, was relocated to make room for new construction while the full complement of 600 employees continued normal operations, including the central telephone switchboard. The Eichleay Company performed this move in two stages, an initial move of 52 feet, and then a rotation of 90°.[4]

Today, telephone switching centers are clearly computer applications. The move of the Indiana Bell Telephone Building was undertaken precisely because an interruption of telephone service while changing buildings was unacceptable. IT-dependent enterprises might differ in the details, but the concept remains unchanged.

[3] "Business as Usual While Moving an Eight-Story Steel-Frame Building," *Engineering News-Record*, July 2, 1931, page 8.
[4] "Business as Usual," p. 4.

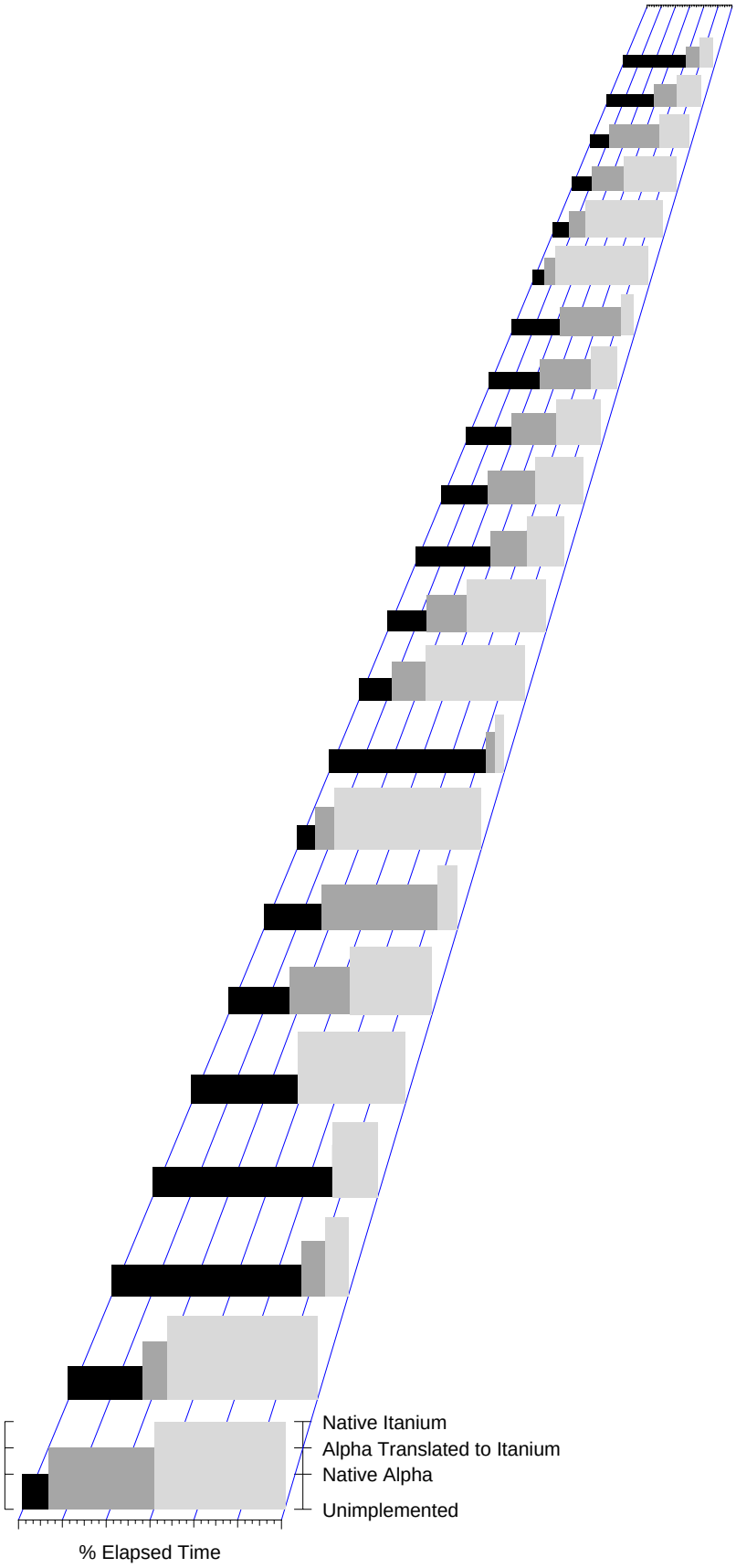


Figure 2 – Each application on an OpenVMS system can be migrated to HP Integrity servers as needed. An initial version might be composed of translated binary Alpha executable images, with the individual component images replaced by native Itanium-based executable images as the native images become available for production use. This can be done on a user-by-user basis or a component-by-component basis. Decoupling of the recompilation and requalification effort from the hardware changeover gives management flexibility in hardware deployments and retirements, without impacting production or functionality. Each software component follows its own transition path from native Alpha image to (optionally) a translated Itanium-based image and, finally, to a native Itanium-based image.

Software qualification and operational commitments

Software qualification is a major hurdle in many enterprises. End-user organizations must qualify software for several reasons, some of which are internal business processes, and some of which are required by external regulations. As compared with ISVs, software rebuilding and requalification are substantially more difficult problems in the end-user environment.

In an ISV or OEM environment, two factors make rebuilding and requalification a simpler problem. First, an ISV or OEM is ultimately in the business of building software. Simply put, this means that the build and qualification process is an integral part of their mainstream business. In contrast, the end-user environment is typically not focused on rebuilding and requalifying software. Another difference is that end-user environments tend to have a much larger number of programs. Although the ISV or OEM can control the scope of the migration, the end-user is faced with the reality of moving hundreds or thousands of applications from one platform to another – truly a daunting task.

Project management and scheduling are also major concerns in an end-user environment. While the ISV or OEM is typically in an engineering environment without day-to-day operational requirements, the end-user must organize rehosting and simultaneously provide uninterrupted support of operational commitments.

OpenVMS provides technologies that allow IT managers to manage change in incremental steps. One example is OpenVMS clusters. When used properly, OpenVMS clusters, particularly mixed-architecture OpenVMS clusters, provide precisely this functionality (Figure 3).

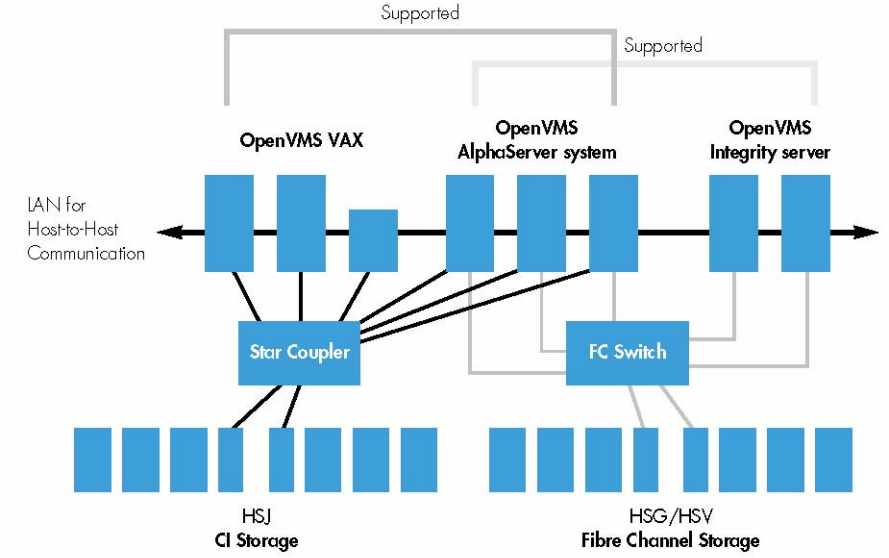


Figure 3 – OpenVMS cluster with VAX, Alpha, and HP Integrity servers (from *The HP OpenVMS Approach to High Availability Computing*, July 2004)

Engineering the details of conversion

The key to successfully moving OpenVMS environments from Alpha systems to Itanium-based systems is understanding and employing OpenVMS facilities that virtually eliminate the risks normally associated with changes in hardware platform.

For simple programs, the task of recompiling, relinking, and requalifying a program is straightforward. The suite of applications software that is at the center of modern data center operations is far more complex than a simple one-page program. The aggregate source base for an

enterprise's locally developed applications programs often can be numbered in the thousands or millions of lines of code. For many application systems, the quality assurance process required of an applications system[5] is a more laborious component than the actual modification and testing of the code itself. ISVs and organizations that regularly rebuild their products are well prepared to perform the steps needed to migrate to Integrity server systems.

The hardware

For much of the last 50 years, migration from one platform to another has been considered a high-risk activity. We sometimes forget that, in the past, changing hardware platforms inevitably involved changing both the hardware platform and the operating system, often involving major changes in both applications programs and the scaffolding of command procedures that surrounded them. Such migrations also involve a "point of no return," after which reverting to the predecessor system is not easily accomplished. Indeed, these migrations, which require significant changes to all aspects of the computing environment, are high risk.

Carefully employed, the OpenVMS environment almost completely eliminates such business risks. Unlike many other hardware transitions, the transition of an OpenVMS Alpha system to an Integrity system is purely a question of changing hardware platform. With rare exception,[6] differences occur only in application programming interfaces (APIs), which are not used by most programmers.

It is important to note that OpenVMS was the first operating system for which a processor architecture was designed, rather than the converse of an operating system designed for a particular processor architecture. This fundamental strength has allowed OpenVMS to assimilate two additional processor architectures, each one radically different from the other.[7]

VAX architecture

The first member of the VAX family, the VAX-11/780, debuted in 1977. The architecture was a byte-oriented, CISC computer with 32-bit arithmetic, 16 registers, integral floating point, virtual memory, and other features. [8]

The VAX-11/780 was the first of a series of systems dubbed "superminicomputers," which provided many features previously available only in high-end mainframes in a price range more familiar to minicomputer users.

Unlike many past computer architectures, the VAX architecture was designed by a combined group of hardware and software engineers. It featured integral support for many software requirements, ranging from instructions for complex array indexing (used by compilers) to instructions for enhancing operating system constructs, such as asynchronous system traps (ASTs).[9]

Alpha architecture

The Alpha architecture, released in 1992,[10] is a 64-bit CPU architecture designed around the concepts of RISC. Originally used in the IBM 801,[11,12] RISC optimizes performance by simplifying the

[5] Levine, Diane E., "Software Development and Quality Assurance," Chapter 25, *Computer Security Handbook,* 4th Edition, Bosworth, 2002.
[6] There are differences in the application programming interfaces (APIs) used to access hardware-specific elements (such as arithmetic fault processing).
[7] The original 32-bit VAX architecture was an example of complex instruction set computing (CISC). In 1992, OpenVMS assimilated the Digital-designed, 64-bit Alpha processor, an example of reduced instruction set computing (RISC). In 2005, OpenVMS Version 8.2 was released, with support for both the Alpha RISC processor and systems based on the Itanium-based Explicitly Parallel Instruction Computing (EPIC) architecture, developed jointly by HP and Intel.
[8] Strecker, W.D., "VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family," *Proceedings of the National Computer Conference,* Strecker,1978.
[9] See *VAX11 Architecture Handbook,* Digital 1978, and *VAX11 Software Handbook* [Digital1978a].
[10] See *Alpha Architecture Reference Manual* [Sites1992].

processor's instruction set. Ideally, each instruction in a RISC CPU requires a single cycle to execute. The Alpha processor has a larger register set (64 registers) than the original VAX but is culturally compatible with its CISC predecessor.[13]

One of the hallmarks of the later VAX processors, and of the entire Alpha series of processors, was an increasing level of hardware-managed parallelism, which was designed to increase the effective speed of the central processor.

Itanium architecture

HP Integrity servers are based on the Itanium architecture, which was jointly developed by HP and Intel. The Itanium instruction set is known as Explicitly Parallel Instruction Computing (EPIC).[14]

The Itanium architecture is different from the VAX and Alpha architectures in a number of ways. The VAX and Alpha designs are characterized by a separation between software optimizations and hardware optimizations. While there was coordination at the design and architectural level, there was no way for compilers on VAX and Alpha systems to signal information for code optimization to the hardware. VAX and Alpha were both designed so that coding in the actual machine code was viable.

This is not the case with the Itanium architecture, which presumes that Itanium machine code is compiler generated. The Itanium architecture removes hardware-imposed conflict resolution for registers. Instead, the compiler code generators are responsible for all such scheduling. The Itanium architecture also has facilities for predication, which reduces the need for branches, and cues for speculation, which indicate the probability of those branches that do exist.

Another difference between the earlier processor architectures used with OpenVMS and the Itanium architecture is the handling of registers on subroutine call. The earlier architectures performed at least a partial register save at each subroutine call. In modern high-level languages with call-intensive paradigms (such as C/C++), this leads to a high number of memory references at each call, resulting in a memory bottleneck. The Itanium architecture implements a register windowing scheme that allows the elimination of these register context operations.

The concepts behind the Itanium architecture appear to be dramatically different from the instruction set architectures that have been the mainstay of computing since the mid-1950s. However, examined in detail, this difference is far less important than it seems. Architectures that are generally similar to EPIC have been part of computing for more than two decades, through the mechanisms used to implement the familiar von Neumann architectures (for example, IBM System/360, DIGITAL PDP-11, VAX, and Alpha).

OpenVMS application programming interfaces

The true test of portability is whether an unchanged source program can be compiled for two different architectures.[15] This has been feasible for OpenVMS users since the debut of VAX/VMS in 1977. OpenVMS has been deliberately implemented on each of the architectures selected so that it is source compatible with its predecessors. It is not uncommon for programmers to successfully use 20-year-old documentation for most of their development work without ill effect. This ability to

---

[11] Cocke, J., and Markstein, V., "The Evolution of RISC Technology at IBM," *IBM Journal of Research and Development*, Volume 34, Number 1, 1990 [Cocke 1990].
[12] Rodin, G., "The 801 Minicomputer," *IBM Journal of Research and Development*, Volume 27, pp. 237–46, 1983, [Rodin 1983].
[13] Dobberpubl, et al., "The MicroVAX 78032 Chip, A 32-bit Microprocessor," *Digital Technical Journal,* Volume 2 (March 1986), pp. 13-14, [Dobberpubl 1986].
[14] Schlansker, M.S., and Rau, B.R., "EPIC Explicitly Parallel Instruction Computing," *Computer* (February 2000) [Schlansker 2000].
[15] Or with minimal changes.

assimilate dramatic changes in underlying technology without changing the API has been one of the historic strengths of the OpenVMS operating system.

For this reason, many programs written for early versions of VAX/VMS continue to run unchanged over 25 years later. In many cases, even recompilation has proven unnecessary. This consistency is not an accident. It is the result of conscious engineering decisions

.

Nonprivileged code

Central to the sequential migration from VAX to Alpha and from Alpha to HP Integrity is the fact the each pair of architectures fully supports the same data types. Thus, with appropriate care, it is possible to perform precisely the same computations on Alpha as on VAX and, in turn, on HP Integrity as on Alpha. Table 1 shows the compatibility of these formats.

	Architecture		
Characteristic	VAX	Alpha	Itanium
Character	ASCII	ASCII	ASCII
Floating-point format	VAX	VAX, IEEE	IEEE
Integer	32-bit	32/64-bit	32/64-bit
Number format		Two's complement	
Address size	32	64	64
Architecture type	CISC	RISC	EPIC

Table 1 - Data formats native to each processor type[16]

Correctness should not be confused with efficiency. When the VAX architecture was originally implemented, memory sizes were dramatically smaller, and packed data structures were considered standard practice. The more recent Alpha and Integrity platforms have dramatically larger basic memory sizes and correspondingly larger performance penalties for accessing unaligned data. Therefore, it is now preferable to use data structures whose elements are aligned on their natural boundaries.

In the past, differing hardware architectures directly impacted many issues visible to programmers, even applications programmers working many levels above the actual hardware. In those days, switching hardware platforms meant changes in integer representations, character sets, floating-point formats, and operating system interfaces. Each of these changes wreaks havoc on an installation's inventory of existing programs.

Today, programmers working at levels above the actual machine language -- even third-generation languages such as Fortran, PL/I, COBOL, BASIC, C/C++) -- are little affected by the mechanics of the platform's instruction set, whether it is CISC, RISC, or EPIC. For that matter, programmers using Macro-32[17] on processors other than VAX are, in effect, using a higher-level language compiler rather than the assembler used on the VAX architecture.

The underlying processor architecture does not become visible until the programmer uses either the option to display the actual generated code (for example, the `/MACHINE_CODE` option in the OpenVMS C/C++ compiler), or the options in the symbolic debugger to display the actual machine-language code. Both of these generally are rare events. It is possible to work for years, or even decades, without exercising either of these options.

[16] Gezelter, R., "The Third Porting: Applying Past Lessons to the Alpha/Itanium Transition," CETS, September 2001 [Gezelter2001].
[17] Macro-32 is the assembler language for the VAX-11/780. See [HPMacro32].

By contrast, the data formats, character codes, and operating system interfaces are directly visible to programmers. On the three architectures supported by OpenVMS, all the available interfaces are the same across VAX, Alpha, and Itanium processors.[18]

Floating-point representations

Floating-point formats are another potential flashpoint. Alpha supports both the original VAX format for floating-point numbers and the IEEE standard representation.[19] The default on Alpha systems is VAX floating-point format. Figure 4 illustrates the differences in the layouts and details of the different floating- point formats.

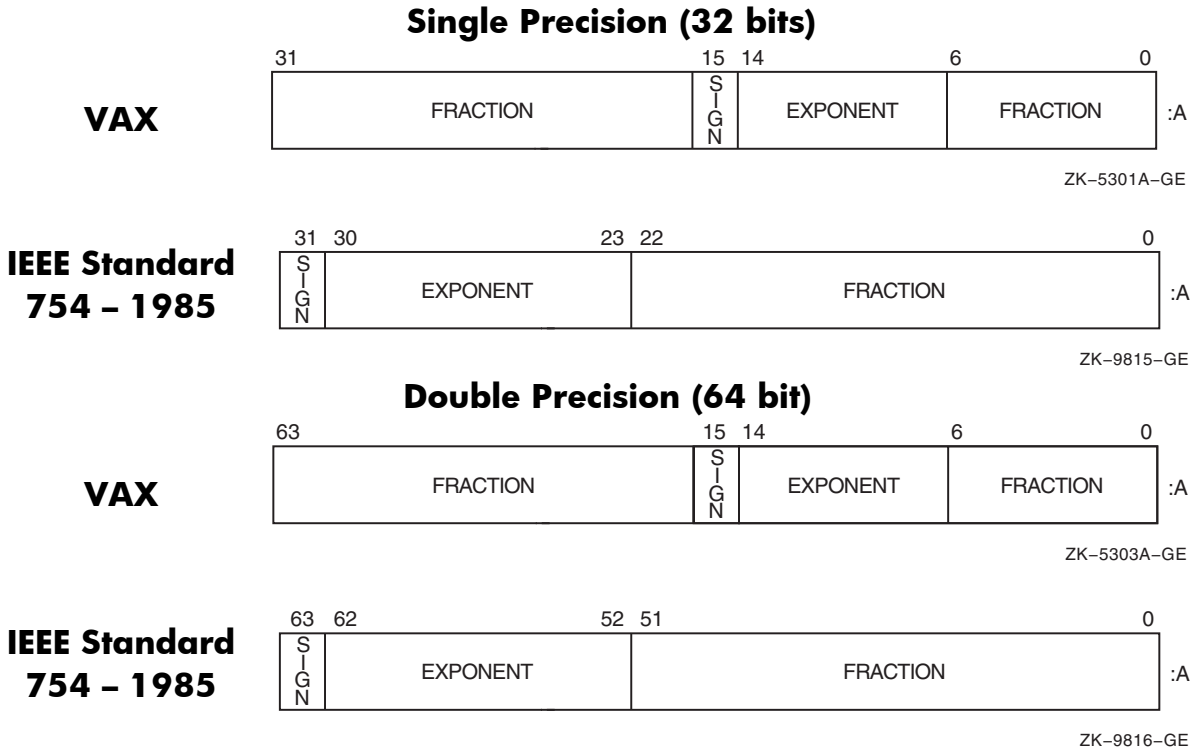


Figure 4 – Single- and double-precision floating-point formats for VAX and IEEE Standard 754-1985 (from *HP FORTRAN for OpenVMS User's Guide*)

Unlike Alpha processors, Itanium processors do not natively support the VAX floating-point formats. For most programs the differences are minor, and for most purposes the results will be the same. The difference in floating-point formats and the resulting small differences in computations is an obstacle in two situations: binary comparison of outputs between Alpha and HP Integrity systems, and some scientific computations.

One standard technique is to compare the output from one system with the output of another system. This technique is often used as a regression test or when migrating code to a different platform. Presumably, if the program produces an identical output for a reasonably sized dataset, it is

[18] Interfaces that are directly used to manage and address the peculiarities of each architecture, such as processing related to processor traps and similar operations, are, of course, different on each of the architectures. The implementation mechanics of subroutine calls are also different on each of the architectures. The mechanics of CALL operations are generally only the responsibility of the various compilers. In the vast majority of code, particularly applications code, these areas are tangential.

[19] *IEEE Standard for Binary Floating Point Arithmetic, ANSI IEEE Standard 754–1985* [IEEE1985].

functioning correctly.[20] For this reason, it is a good idea to recompile and retest programs using IEEE floating point on Alpha as a first step. Once the results of the program are determined to be correct, the same program on an Integrity system should produce the same results.[21]

Privileged code

Programs operating at inner modes, particularly programs that directly manage processor components, are inherently more dependent on the differences between the architectures. These differences matter to developers working at inner access modes and employing hardware-specific features.

Privileged code that is dependent on OpenVMS privileges are generally unaffected by changes in architecture. Such code does not rely upon operations that directly affect the processor or other hardware. As an example, a program that enters kernel mode to call a system component, but that does not otherwise affect processor hardware, is likely to be unaffected by changes in the underlying architecture.

Hardware-specific functionality

At the device and processor levels, the three architectures differ from each other just as processors and systems differ within each of the individual architectural families. These differences are generally invisible, particularly in end-user environments.

Conversion options

In a conventional, non-OpenVMS environment, there is little choice but to do a so-called "cold turkey" cutover of applications from one processor and operating system to another. Such a cutover requires migration of data en masse from one system to another, often requiring reformatting of data for the new system. This is an all-or-nothing approach. Once the process starts, there is often no way to simply reverse the process if something goes awry.

This approach is heavily laden with risk. In today's environment, down time is prohibitively costly to the organization. A study by TechWise[22] showed that even modest down time frequently costs in excess of $10,000 USD/hour. Major system conversions that have gone awry have even impacted publicly reported profits.

The facilities of OpenVMS provide a uniquely malleable environment for changing hardware platforms. Specifically, many strategic and tactical choices are available that allow decoupling of the hardware cutover from the actual applications migration.

An OpenVMS migration from one processor architecture to another is a far lower-risk scenario. Disk data formats are consistent across OpenVMS systems running on different architectures. In fact, OpenVMS clusters provide the mechanisms to decouple data migration from application migration, whether or not an installation uses a Storage Area Network (SAN). Properly managed, it is quite possible to grow OpenVMS storage environments significantly and to change their physical realization without ever interrupting user access to information.[23]

---

[20] Or at least as correctly as the original program was functioning. This technique is popular for writing compilers because it produces a high degree of assurance that the compiler can be used to compile itself. Therefore it can be used to compile its own bug fixes.

[21] In fact, the Java® standard requires the use of the IEEE floating-point formats for precisely this reason. "4.2.3 Floating Point Types and Values", p. 33 [Gosling1996].

[22] TechWise Research, Inc., *Quantifying the Value of Availability: A Detailed Comparison of Four Different RISC-Based Cluster Solutions Designed to Provide High Availability*, Version 1.1a (June 2000) [TechWise2000].

[23] See Gezelter, R., "Migrating OpenVMS Storage Environments without Interruption/Disruption," HP Technology Forum, 2005 [Gezelter2005].

Managing change – the OpenVMS approach

OpenVMS, with its support for mixed-architecture OpenVMS clusters and its provisions for binary translation,[24] provides a way to assimilate Itanium-based servers into a production environment without requiring a "cold turkey" transition.

The combination of image translation and mixed-architecture OpenVMS clusters allows computing capacity and hardware inventory to be transitioned in the same orderly fashion as the software inventory. Computing capacity can be transitioned from one architecture to another as dictated by the changing needs of the applications mix (Figure 5). This allows cost-effective management of the hardware inventory separate from the tempo of the migration effort. Such a strategy allows users to be migrated to the new architecture in a seamless manner, thereby optimizing financial, technical, and support expenses and risks.

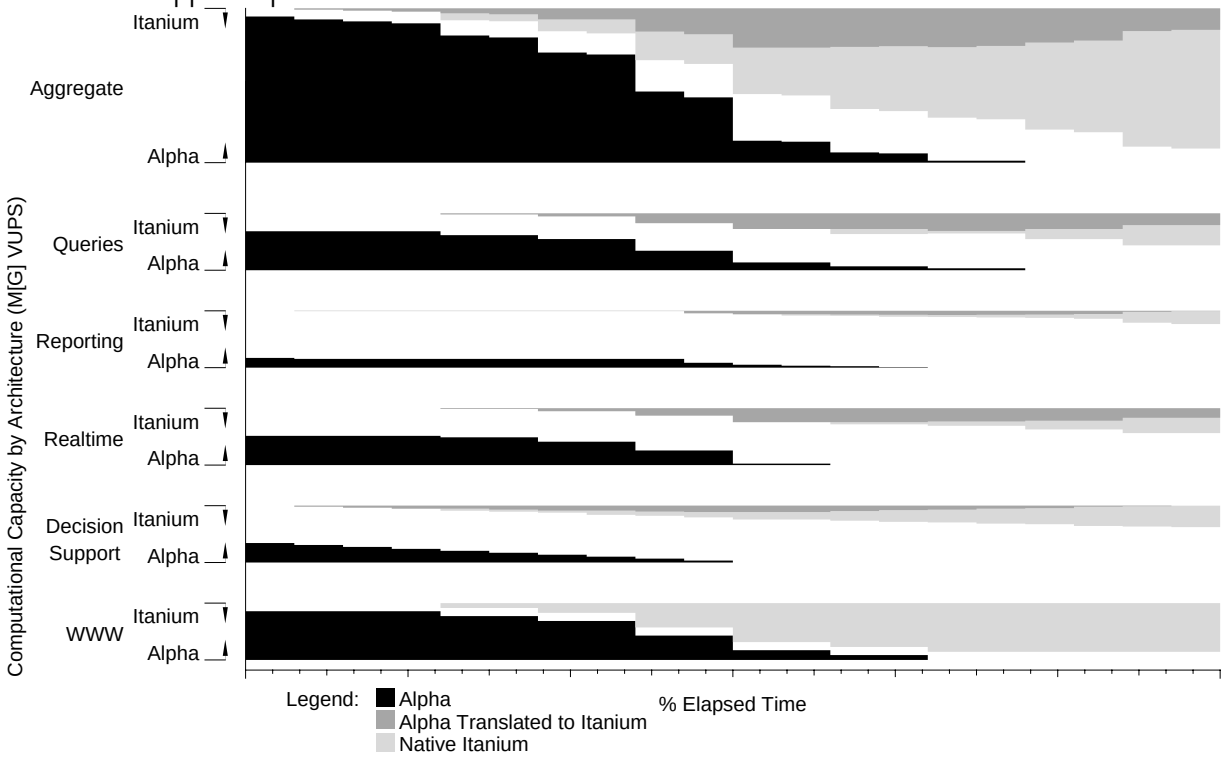


Figure 5 –Mixed-architecture OpenVMS clusters capacity transition from Alpha to HP Integrity servers over time.

Conventional changes in processor architecture give rise to another problem: interdependencies between applications and underlying software components. Modern software applications are not freestanding entities. Each one depends on an extensive collection of supporting software, each of which might depend on other software components. These interdependent components are often maintained by different groups, each with their own priorities, responsibilities, and commitments. Adding to the complexity, some components might be developed and maintained in house, while others might be licensed products.

These interdependencies, which can be devilishly difficult to ferret out, have been the cause of more than one migration setback. Concern and respect for the complexity of this problem has been part of the OpenVMS engineering philosophy since the initial design of the VAX-11/780. These concerns are at the heart of why such transitions are qualitatively safer on OpenVMS.

[24] On Alpha, for code translated from binary VAX executables; on Itanium, for code translated from Alpha executables (which, in turn, might have been translated from VAX binary images).

The solution to this problem of interdependencies was to implement many noncore parts of the operating system and its utilities as PDP-11 images, running under the RSX-11 Applications Migration Executive (AME).[25] This allowed engineering resources to be concentrated on components that had to operate in native mode (for example, the executive, RMS). It also allowed development of supporting components (such as compilers, editors, and tools) on existing, readily available PDP-11 systems. This approach allowed the OpenVMS Engineering organization to ship the first VAX-11/780 in October 1977, only 18 months after finalizing the initial design. The tools and utilities running under the emulation mode were gradually replaced in subsequent releases of the VAX/VMS operating system.[26] The common data formats and character sets between the different architectures mean that data translation was and is rarely required.[27]

In fact, the image-translation support allows native Itanium binaries to invoke translated images, and vice versa. It is such a fundamental part of the operating system implementation that it happens without conscious thought on the part of most users and system managers. Translated images, through transfer vectors, invoke entry points in the normal common run-time library when required.

Translated images have been part of OpenVMS production systems since the advent of Alpha and VAX. Perhaps the most commonly used translated component in recent history has been the Monitor utility (MONITOR) on the Alpha platform. Unheralded and virtually unnoticed, the Alpha version of `MONITOR` has been translated from successive VAX images since the advent of the Alpha in 1992 through 2005.[28] Similarly, the `TECO` text editor has made use of emulation and translation since the advent of the VAX in 1977.

The preceding examples suggest a prudent, time-tested strategy -- one that has been used three times by the OpenVMS Engineering organization itself to move a development and production environment from one hardware architecture to another without interruption.[29] This strategy is characterized by an approach that dramatically reduces the Gordian knot[30] of components that must be migrated in order to have an operational production system.

Environments have a range of different applications (Figure 6). Core applications are critical to daily operations; other applications might be critical, but on a far longer time scale. Still others might be important but are not used on a regular basis.

---

[25] [Digital1978a], pp. 11-1 – 11-8.

[26] The last mainstream component to run in emulation mode was the SOS editor, retired eight years later with release of VAX/VMS Version 4.0. This coincided with the release of the MicroVAX I, the first VAX-family processor that did not include native hardware support for the PDP-11 instruction set. At that time, the AME, now a separate product, was enhanced with the addition of software instruction set emulation, allowing customers to continue to run PDP-11 images on VAX systems.

[27] The data types used in VAX translated images are available on Alpha. The data types used on Integrity are common with Alpha. All of the data types, with the exception of the floating point formats, are the same across all three architectures. The Alpha Environment Software Translator (AEST) has provisions for calling software routines to process VAX floating-point formats transparently, albeit with some loss of efficiency.

[28] The `MONITOR` utility on Alpha, through OpenVMS Version 7.3-2 – a period of over a decade – was translated from the VAX executable by the VAX-to-Alpha binary translator. The majority of the user community was not aware of this difference until the native mode MONITOR was announced with the release of Version 7.3-2.

[29] PDP-11 to VAX (1977 et seq.); VAX to Alpha (1992, et seq.); and Alpha to Integrity (2005, et seq.)

[30] The Gordian knot was a complicated knot that could not be untied. It was cut by Alexander the Great.

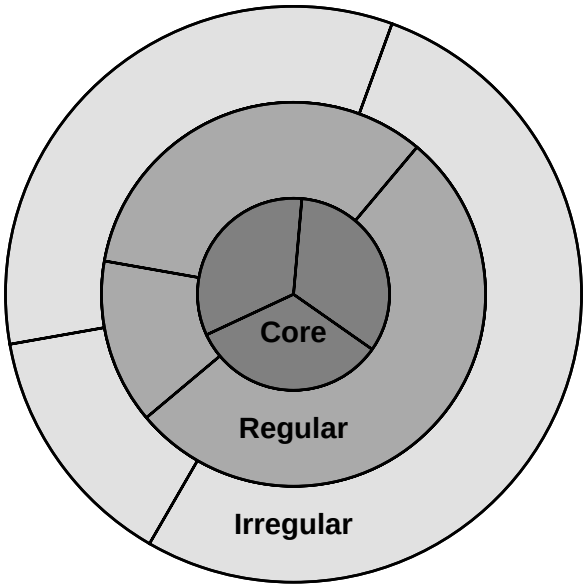
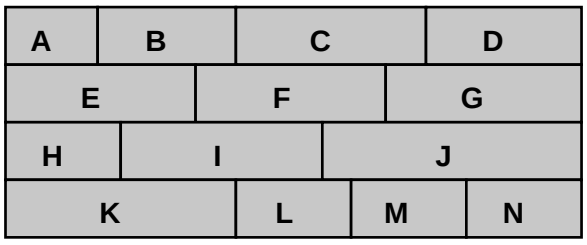


Figure 6 – Typical environments include a range of applications, from core to irregular utilities and reports.

Each end-user environment is unique and has its own issues — technical, political, and financial. In some environments, it is desirable to migrate core applications first. In others, the best candidates for early migration are the rarely used utilities that are not critical to day-to-day operations. Even two organizations in similar niches in the same industry can have different preferences for the order in which workload is moved from the old platform to the new platform. One of the strengths of OpenVMS is that this decision, with all its nuances, is driven by end-user business issues on a case-by-case basis.

Workload migration is also not limited to application-by-application migration. OpenVMS clusters might include systems with differing architectures transparently accessing the same files. Once an application is believed to be ready for production, it is possible to rehost the user community in stages that are calculated to minimize risk and strain on user support resources. Thus, in a well-executed OpenVMS migration, few situations require the transition to be made all at once.[31]

This capability of mixing and matching components allows a wide degree of flexibility when scheduling transitions of components from translated to native compiled forms (Figure 7). Initially, production might use an application entirely composed of translated images. As images become qualified for production use, it is a simple matter to integrate them into the production profiles. When this iterative process is completed, all of the images are natively compiled for the Itanium architecture.



[31] That is, changes made purely to the architecture of the underlying system. Pure changes to the hardware realization of resources can also be accomplished without disruption. Some changes, such as overall changes to the security scheme, might require "cold turkey" transitions.

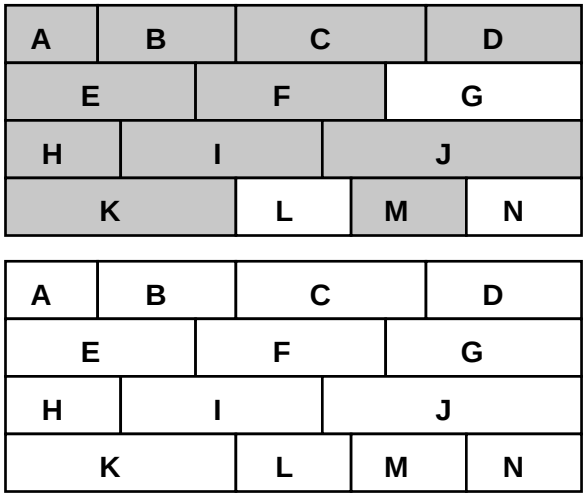


Figure 7 – Translated images allow the migration to be achieved one component at time.

Mixed-architecture clusters, together with binary translation, allow workloads to be moved to newer platforms in an orderly fashion that is compatible with all the requirements of business: financial, operational commitments, and technical resource availability. The same flexibility that is available with different processes is also usable on a shareable library-by-shareable library basis. Applications rarely exist as singletons. In the majority of cases, end-user environments have many applications, often grouped into families that use common supporting libraries.

The translated-image facilities enable individual libraries to be used in translated or native environments, as appropriate. Additionally, since the choice of library is controlled through the use of logical names,[32] different libraries (one native, one translated) can be used on the same cluster member, depending on some combination of user, application, and other criteria.[33]

Compiling natively for the Itanium architecture

Actively maintained applications are far easier to migrate. Active maintenance means that the code base[34] is recompiled and relinked regularly. Current code bases are less likely to use obsolescent or variant language features that are not available on the Itanium compilers. In the best case, these code bases can be migrated quickly to a native Itanium environment simply by recompiling, relinking, and requalifying.

In many environments, this smooth state of affairs is not the case. It is common to find executable images and shareable images that have not been rebuilt or recompiled for over 10 years. It is also common to find a fairly small staff responsible for literally hundreds of programs. In such an environment, the sheer volume of code makes even modest changes a daunting, drawn-out project.

Recompiling

Theoretically, recompilation is a routine process. The challenge is caused by the changes in the underlying language or options that are supported by the compiler. These can be extensions

[32] Section 8.3.2 *Activation of Shareable Images*, pp 336 [Goldenberg1997]
[33] Gezelter, R., "Inheritance Based Environments for OpenVMS Systems and OpenVMS Clusters," *OpenVMS Technical Journal*, Volume 3, February 2004 [Gezelter2004].
[34] A code base is the collection of source files that are compiled to produce an image. This includes source, header, include, and similar files. External libraries used by the resulting image are not generally considered part of the code base for an individual application.

available with the VAX or Alpha compilers that are not available on Itanium-based systems, [35] or they can be intervening changes in the level of the language.[36] In either case, source changes must be made before the program can be recompiled, and the recompiled program must be qualified to ensure that the changes have been made correctly.

Recompilation on Itanium-based systems also changes the default floating-point format from VAX to IEEE, except in the case of Java, whose specification already requires the use of IEEE floating point.

Linking

At the user level, there are few differences when linking on any of the architectures supported by OpenVMS.[37] Each architecture has its own object and executable file formats. On HP Integrity systems, the System V Executable and Linkable Format (ELF) is used for object files, shareable images, and executable images.[38]
The different formats are not an issue for users. Compilers targeting each architecture generate the correct object format for that architecture, and the Linker processes these object modules to create executable image files. Linker command files used on Alpha systems should not need changes to operate correctly on HP Integrity systems.

Translation versus emulation

Translation is far different from emulation. Emulation reinterprets the actual binary image of the instructions used by an earlier (or different) architecture. In the classic OpenVMS example, the original VAX-11 series processors[39] had integral hardware to emulate the nonprivileged aspects of the PDP-11[40] instruction set. Later VAX products used software emulation, which allowed them to execute unchanged binary images from RSX-11M/M-PLUS systems.[41]

The penalty for the flexibility of emulation is that each instruction must be reinterpreted each time it is used. Depending on a variety of factors,[42] software emulation produces extra overhead that reduces performance, often by a factor of 5 to 10.

Translation is inherently different. Translation uses an offline utility, the translator, to convert a binary executable image from the instruction set and format of one architecture to the instruction set and format of a different architecture. This process is akin to using a compiler in that it is run a single time. Thereafter, the resulting image executes without extensive supervision.[43]

Performance

The performance of translated binary images depends greatly on what actual processing is performed. Later in this article, some of the case studies show how the OpenVMS shareable library

---

[35] As an example, the OpenVMS C compiler supports the non-ANSI C Language extensions provided by the earlier VAX C compiler. The C compiler for Itanium-based processors does not support these extensions. The changes to bring code into compliance are not large, but in a large source base the sheer scale of the effort is a challenge.
[36] The Fortran compiler for Itanium-based processors supports Fortran-90. Many programs were written using earlier versions of the Fortran standard.
[37] VAX uses a different mechanism for declaring entry points to shareable libraries. Additionally, support related to 64-bit addresses is limited to the Alpha and Itanium implementations.
[38] See [TIS1995].
[39] VAX-11/780, VAX-11/750, VAX-11/730, VAX-11/782, VAX-11/785, VAX-11/725, VAX 8600, and VAX 8650.
[40] See [Digital1978c].
[41] Because the operating system API for the RSX-11 family differed from that provided by OpenVMS, a software library referred to as the Applications Migration Executive (AME), provided the mapping between the RSX-11 executive directives and the system service calls used by OpenVMS.
[42] See "Binary Translation," *Digital Technical Journal,* Volume 4, Number 4, 1992 [Sites1992a].
[43] The Translated Image Environment (TIE) for Alpha images executing on an Itanium processor includes an Alpha instruction set emulator. This emulator is invoked automatically when a fault caused by an attempt to transfer control to a block of instructions not identified as executable code by the binary-image translation utility, the Alpha Environment Software Translator (AEST). Therefore, each translated Alpha executable image contains a complete copy of the original Alpha executable.

facility and the image translation utility can be used in concert to optimize the timeline. This optimization is achieved by concentrating efforts where the most performance can be gained, and by translating other parts of the application that are not as performance sensitive.

Appropriate choices

Modern business must be agile, quickly adapting to changes in the business environment. This need occurs at all levels, including business strategy, IT provisioning, and IT operations. In IT provisioning, OpenVMS provides developers and maintainers with a range of choices for how to proceed with migration to a new hardware platform. In the simplest case, with plentiful resources and fully compliant sources, it is simply a matter of recompiling, rebuilding, and requalifying. This is generally the choice of software developers and of those with isolated programs. The problem of large, interrelated applications environments is a different world. Here, the conflicting requirements of different obligations and costs can easily create a situation of tangled interdependencies.

When resources or schedules are less favorable, OpenVMS features provide the ability to divide the migration into manageable, low-risk segments (Figure 8). These steps can be performed incrementally, with the end result of total conversion to native operation on Integrity servers, but without the risks associated with an instantaneous cutover.

Users are often required to log out of applications at various times in the work day (for example, at breaks, at meals, or at end of shift). These natural operational boundaries can be harnessed as points at which users, either as individuals or as groups, can be transitioned to changes in the underlying environment. Replacements for each component or group of components can be accomplished at the proper time, as a side effect of routine operations. This approach results in an orderly transition that does not impact users.

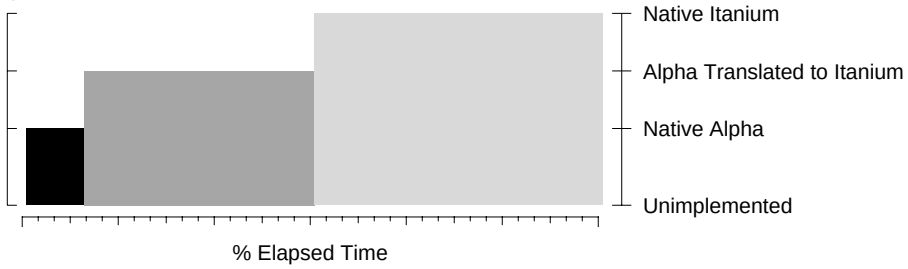


Figure 8 – An individual application or shareable image component of an application can be transitioned from Alpha to HP Integrity systems in stages, from image translation to native execution. This allows users to be transitioned from Alpha servers to HP Integrity servers without the need to recompile code.

Mixed-architecture OpenVMS clusters can be used in the same fashion. The ideal of transferring an OpenVMS cluster composed of VAX and Alpha nodes and affiliated storage to a pure OpenVMS cluster with only HP Integrity systems and current-generation storage arrays is achievable. When done carefully, it can be accomplished without any total interruptions of overall system availability. No changes to production routines other than the logins and logouts routinely done during the workday are required.

The procedures of cutting over OpenVMS cluster members without shutting down the entire cluster are well understood, as are the techniques that enable the migration of storage without interrupting user or application access.[44] The following sections focus on the software management steps that allow these goals to be achieved.

[44] Gezelter, R., "Migrating OpenVMS Storage Environments without Interruption/Disruption", HP Technology Forum 2005, Orlando, Florida, October 20, 2005 [Gezelter2005].

For example, it is not generally appreciated that, until the release of OpenVMS Alpha Version 7.3-2 in 2005, the Monitor utility, which monitors system performance, was a binary translation of the image used on the VAX version of OpenVMS. The entire OpenVMS universe has been using `MONITOR` and other translated images on Alpha for more than a decade without incident or, for that matter, awareness.[45] The same technology alternative is available for translating Alpha binary images for use on HP Integrity servers.

Five-step migration process

As shown earlier, there are many options for migrating a base of applications from an older OpenVMS architecture to the Itanium architecture. The important issue is how to accomplish the technical migration with minimal risk and maximal efficiency.

The steps in the transition process are:

Step 1	Establishing automatic qualification procedures.
Step 2	Translating all binary images to Itanium images; requalifying.
Step 3	Normalizing data formats.
Step 4	Updating source files to latest version of OpenVMS (Version 7.3 on VAX, Version 8.2 on Alpha); requalifying.
Step 5	Recompiling sources for HP Integrity, producing native Itanium images; requalifying.

Step 1 – Establishing qualification procedures

During the transition process, the team must verify proper functioning of the applications multiple times. Standardizing the qualification procedures for each application ensures that these tests are done in a consistent manner at each point.

These tests fall into two categories: functionality tests and regression tests. Tests of functionality guarantee that software can perform its intended functions. Regression tests are a far different category. The collection of regression tests for a program is the collection of test cases for problems that have been resolved. Running these tests as part of the qualification process ensures that problems do not reappear once they are corrected.

OpenVMS features such as the Logical Disk driver (`LDDRV`),[46] the Backup utility, and the logical name facility make it straightforward to implement re-creatable test environments with low risk and a high degree of reproducibility.

Step 2 – Translating binary images and requalifying

En masse binary translation of all images at this point, followed by requalification, provides an assurance that the application as a whole can be run on HP Integrity servers.

An application is often not a single image. It is common for a business application to consist of many images, some directly executable, others used as shareable libraries. Creating translated, qualified versions of these images ensures that developers, each working on their own part of the HP Integrity server system, can be assured that the software surrounding their individual piece is sound. It is not necessary to use the translated software in end-user production, although as `MONITOR` has demonstrated, it is an appropriate option.[47]

---

[45] There are often concerns about introducing new technologies during an architectural transition. The fact that the technology has been in widespread, everyday use is often a source of re-assurance.

[46] Originally (and still available as) freeware, LD is integrated with the standard OpenVMS distribution as of Release 8.2-1. See [vanderBurg2005].

[47] The choice depends upon the economic and hardware issues referenced earlier in this paper.

Step 3 – Updating source files to current language level

Many programs have been run for years or decades without recompilation. In the intervening time, language standards have changed. For example, many users have source programs that date to the Fortran-66 or VAX C standards.

Compiling these programs with current-generation compilers often results in errors and warnings. It is important to resolve these issues before attempting to transition the software. This step isolates these changes in compilers from changes brought on by the new architecture.

Step 4 – Normalizing data formats

The default[48] floating-point formats on Alpha have historically been the floating-point formats used on the VAX[49] and PDP-11.[50] The IEEE floating-point standard is supported by the Alpha compilers and is the default on HP Integrity servers. The results produced by equivalent computations using each of the two formats differ slightly, generally in an insignificant way.[51] For this reason, the format change is listed as a separate step. The other steps should produce the identical results on a bit-for-bit basis.

Step 5 – Recompiling sources for HP Integrity systems and requalifying

Once the source base is up to current language standards, and no issues have been identified relating to the change in floating-point format from VAX to IEEE, the source base can be compiled natively on the HP Integrity system.

Rationale

It might appear that the translation and recompilation steps are redundant, but this is far from the case. Some locally developed applications might consist of a single executable with a small source base. However, an application environment consisting of hundreds or thousands of executable images is much different. The dependency diagram of such an application collection illustrates why the multiple stages are needed. Simply put, at the end of step 4, we can be certain that the majority of older hardware can be retired.[52] This development disconnects the hardware retirement from total conversion of the applications base.

Example scenario

This scenario is one of multiple, interwoven applications and libraries, where the network of interdependencies virtually guarantees conflicts and problems. The complexity of each case study illustrates choices in the process that would not be apparent in simpler situations.

The case involves a large data-processing organization with a number of semiautonomous development groups, each responsible for some segment of the applications that support the overall business. The organization might be a financial institution, a manufacturing operation, a utility, a research project, or some other enterprise.

---

[48] Java is the exception. Compliance with the Java specification requires the use of IEEE floating-point format. It is possible to support Java calling of external routines using VAX floating point, but a small jacket transfer vector is required.

[49] See [Digital1978].

[50] See [Digital1978c].

[51] Complex numerical calculations (such as fluid dynamics, Monte Carlo methods) take into account and depend upon the behavior of the lowest-order bits of the computational results. These computations might be affected by the change in floating-point formats.

[52] Until it is possible to successfully natively recompile and requalify *all* of the executable images, it is unwise to lose the ability to recompile on the original platform, whether VAX or Alpha. Two options for retaining this capacity are available: workstations and emulators. Workstations can be an inexpensive hardware option to retaining the ability to compile and test VAX or Alpha programs in their native architectures. A second option is the use of a VAX or Alpha emulator, such as the Charon series of emulators available from Software Resources International (`http://www.softresint.com`).

The hardware environment is currently a moderate to large cluster of AlphaServer systems. Much of the hardware is leased, and the leases expire in a relatively short time. A large number of the applications are developed in house. The staff is confident that they can perform the transition, given the time to do the work of upgrading many old sources, which have been in use for years without recompilation. Many of the images depend on a series of underlying shareable images that are maintained by several of the development groups.

Management is concerned about finances and availability. A brief review of the finances, personnel, commitments, and dependencies shows a complex series of connections which, at first glance, seems irresolvable. In short, there appears to be no way, without dramatic increments in budget and personnel, that the transition can be accomplished without extending the leases on the existing AlphaServer systems, the cost of which is prohibitive.

This is not a far-fetched scenario.[53] What is unusual is that image translation, mixed-architecture OpenVMS clusters, and host-based volume shadowing provide the enabling technologies to transition this admittedly complex environment without ever interrupting production use of the system.[54] If problems do occur, the transition can be undone in seconds or minutes, at any point in the process.

The first two steps, which can be done somewhat in parallel, are to integrate one or two Integrity servers into the existing OpenVMS cluster, with access to all of the required data volumes, and to develop qualification criteria for the various components which comprise the applications base.

The next step, image translation, is a brute force process. Each image, whether directly executable or shareable, is translated for use on Integrity servers and then requalified. The end result of this process is a full applications environment running on the Integrity platform.

How the resulting translated environment is employed is a business decision that depends primarily on cost and risk. The environment can be used as one or both of the following:

- Allow the different development teams to work in parallel, each on its own sections of the applications base, using the translated images as a framework or scaffold.
- Shift production use to the Integrity platform using the translated images, redoing the translation process for images affected by patches.

This is not an all-or-nothing decision. The translated environments can be used for production use in some applications and not in others. In many environments, the performance of the translated images is adequate for transitional production use and possibly for long-term use.[55]

Once the translation has been qualified, it is possible to retire and remove the majority of the AlphaServer systems from the OpenVMS cluster and replace their computing capacity with HP Integrity servers, as required. It is recommended that at least one AlphaStation be retained as a resource for the interim rebuilding of executable images for translation, but that is a nominal expense.

The development teams can now work independently on the language and other minor changes needed to natively recompile their components on Integrity systems. As each image is recompiled and qualified for native use on HP Integrity systems, it can replace its translated version without disruption. At all times, the full functionality of the system is available to users and developers.

[53] In fact, this is essentially the scenario addressed successfully by the OpenVMS Engineering organization three times: initial release, the transition to Alpha, and the transition to Integrity servers.
[54] Assumes that the user data is already stored on volumes configured as extendable shadow set members under host-based volume shadowing.
[55] The OpenVMS Monitor utility is a living example of the utility of this approach to technical management.

Operational issues

Operational commitments greatly complicate the situation. Cold cutovers are an acceptable strategic approach only if extended interruptions in system availability are not an issue. The recommended approach is a phased, incremental cutover. If a problem occurs while moving a group of users to the newer version of the application, it is a straightforward matter to revert to the earlier application and undertake analysis of what transpired.

Case study 1: A simple case of a common library

The customer application is one step more complex than a simple, self-contained program. Each of the five applications in this case study uses a common set of underlying utility routines that are coded in Macro-32 (the assembler language for the VAX-11/780) and ANSI C.

The first step is to rebuild the original application on the Alpha by using a shareable library form of the utility routines rather than a set of object modules statically included at link time.

The applications are maintained by a different staff than the maintainers of the underlying utility library. Each group has its own schedules, budgets, and commitments. The goal is to transition this code base, which was originally designed for a VAX system more than 15 years ago, into a form that executes natively (without translation) on the HP Integrity platform. Of prime importance is that the transition to HP Integrity systems does not affect the use of the application for production on a daily basis.

The first substantive step is for the two groups to run their respective images through the Alpha Environment Software Translator (AEST), and to verify that the translated images function correctly. This step serves three purposes:

- To provide a baseline of functionality for the converted images.
- To permit production use to be transitioned to Integrity servers while the actual migration work is underway.
- To keep the system as a whole available throughout the course of the source modifications, thereby allowing both development groups to work in parallel without cross-interference.[56]

First, the utility library is translated using the AEST command:

```
$ AEST UTILITY_LIBRARY
```

Next, each application that is linked against the library is translated:

```
$ AEST APPLICATION1
$ AEST APPLICATION2
$ AEST APPLICATION3
$ AEST APPLICATION4
$ AEST APPLICATION5
```

The translated versions of both the library and the applications mutually reinforce the abilities of their developers to migrate the entire environment to the HP Integrity servers. It is also possible for the native versions of the executables to be integrated into the production environment on a step-by-step basis. If necessary, this integration can be accomplished on a group-by-group or individual basis.[57]

[56] Application developers can use the translated version of the utility routines to support their migration and testing. Utility developers can use the translated versions of the applications as the necessary foils for the native version of the utility library.

[57] The use of group-specific or job-specific logical names provides the foundation of this capability. See [Gezelter2004].

Case study 2: A large, well-maintained source collection

C-Kermit[58] is a good example of a program with a long, productive life. C-Kermit has been through several versions (Version 8.0 being the latest) and has been maintained by a loose confederation of developers. As an example, C-Kermit consists of a collection of source files, as shown in Table 2.

	Files	Lines	Blocks
*.c	41	277,376	18,327
*.h	22	17,409	1,018
Total	63	294,785	19,345

Table 2 - C-Kermit Version 8.0 Source Base

A casual review of the sources reveals a common problem with older source collections: the extensive use of variables with global scope.[59] These pervasive global variables make it difficult to divide the program into a series of self-contained shareable libraries. Fortunately for the C-Kermit maintainers, the conventions of ANSI C are well enforced, and the source was relatively straightforward to natively recompile for the Itanium architecture.

Suppose the source base had not been strictly ANSI C, and quick recompilation was not an option. Would translation have resulted in a production-usable utility?

A straightforward binary translation of the Alpha executable into an Itanium executable was done as an experiment. The performance of that executable was approximately 50% of the performance of a native compiled image (0.29 seconds versus 0.11 seconds; see Table 3) for the transfer of the distribution of INFO-ZIP (6,874 blocks); but that is quite acceptable.

Case study 3: Multiple, interdependent underlying libraries

Case study 1 illustrates the ease with which this approach decouples the physical aspects of the migration from the software engineering effort. The case of C-Kermit (case study 2) is a relatively simple one and almost does not seem worth the effort. Expanding case study 1 to include a larger set of libraries and multiple applications, with varying sets of the libraries begins to illustrate the complexities that can occur.

The apparent simplicity of case study 1 becomes a far more complex scheduling problem when one considers these issues.

More commonly, large applications environments are far more complex with many more interconnections and interdependencies.

Case study 4: A gradual transition of elements

Some applications contain single executable images whose source modules comprise aggregates of tens of thousands of lines in one or more languages. These applications are often maintained by teams of maintainers. The challenge in such a situation is to find a task sequence which allows all of the teams to meet their objectives.

The combination of the image translation facility, the Translated Image Environment, and the user-by-user (or finer) control over which shareable library is used uniquely provide significant leverage for

[58] Maintained by the Kermit Project located at Columbia University. See `http://www.columbia.edu/kermit`.

[59] Variables with a global scope are defined in C/C++ outside the scope of a single procedure. Within the compilation unit, they are visible to all modules. Outside of the compilation unit, they are referenced using the *extern* keyword. The references are then resolved by the Linker utility. Fortran, COBOL, and other languages have mechanisms with similar effects but different syntax.

OpenVMS users in managing and controlling this process. An example from a recent client project illustrates the mechanics of how this can be accomplished.

The `ITEMCHECK` program was implemented to cross-check and validate data contained in a series of RMS files containing data from a client relational database. Each RMS file contains data[60] from one table of a PC-resident relational database.

The sources needed to build `ITEMCHECK` are:

- The actual source for the `ITEMCHECK` program
- Some library collections developed for this particular project
- Some library collections developed for other projects

The individual libraries were implemented to be independent of one another. While each library does make use of variables with global scope, the variables are referenced only from within each library. Some interfaces, reminiscent of the C++ object-oriented programming style, are used to set and get these variables when necessary. Other static variables are used internally within libraries and routines and are not exposed directly to outside interfaces.

It is a simple matter to break the different components into a main image and eight shareable images. Each of the images can then be translated individually. The source base for `ITEMCHECK` and its underlying utilities is not particularly large, but it is large enough to be illustrative.

```
$ AEST ITEMCHECK
```

Each shareable image can now be translated individually:

```
$ AEST ITEMCHECK_SHRLIB
$ AEST DEBUGTRACESHR
$ AEST GRAPHICSSHR
$ AEST INDEXSHR
$ AEST ITEMSHR
$ AEST ORDERSHR
$ AEST OUTLINESHR
$ AEST PRODUCTSHR
$ AEST READKEYEDSHR
$ AEST RMSSEQUENTIALSHR
$ AEST UTILITIESSHR
```

The resulting collection of shareable images can be executed on an HP Integrity server without native recompilation. On an individual basis, each library can be transitioned from its translated version to its natively compiled version, if need be, on a user-by-user or application-by-application basis, until the entire assemblage has been compiled natively and fully qualified.

On a project basis, this provides a large number of alternative paths for project management.[61] Thus, a wide variety of benchmarks can be run in this case.[62] A sampling of these benchmarks (Table 3) shows the performance of this program when processing a representative dataset containing 1,000,000 line items comprising 500,000 orders.

---

[60] Because of nondisclosure agreements, the low-level details of the code cannot be released. The performance numbers contained in this paper represent runs with synthetic data comparable in character to the actual client data.

[61] Since there are a total of nine images (one main image and eight shareable libraries), there are a total of 512 (2^9) different combinations. This represents a dramatic increase in management flexibility and project scheduling.

[62] To be precise, 516: 512 combinations of the translated/natively compiled shareable libraries, the shareable image on Integrity servers without support for translated images, the single native image on Integrity, the original single native image on Alpha, and the shareable image component on Alpha.

AlphaStation 200 4/233; 384 MB, 2 RZ-29				SPECint92: 157.7	
	CPU	Buffered IO	Direct IO	Virtual Size	Page Faults
Tare	0.16	58	8	171,584	173
C-Kermit 8.0 (Client side)[63]	17.73	19,327	40	184,512	749
ITEMCHECK					
Single Image	2,305.88	167	59,551	175,200	462
Shareable Images	2,462.81	197	59,582	175,200	490
Integrity rx2600; 1.4 GHz/1.5MB Cache; 2 GB; 2 36G				SPECint2000 1170+[64]	
	CPU	Buffered IO	Direct IO	Virtual Size	Page Faults
Tare	0.01	48	15	176,624	180
C-Kermit 8.0 (Client side) [65]					
Translated	0.29	19,868	57	198,288	764
Native	0.11	20,240	48	198,288	764
ITEMCHECK					
Single Translated Image	5,978.47	147	59,500	198,944	714
Translated Shareable Images	3,501.47	170	59,497	201,264	805
Assortments of Native/Translated Images					
Translated Main Program					
Mask 194	735.37	173	59,599	198,944	752
Mask 38	2,262.38	182	59,501	198,944	753
Mask 9	1,365.09	175	59,500	198,944	782
Native Main Program					
Mask194	179.41	180	59,501	198,944	750
Mask 38	2,016.53	177	59,500	198,944	752
Mask 9	823.49	170	59,499	198,944	779
Native Shareable Images	175.49	151	59,491	198,944	570
Single Native Image	163.01	121	59,482	198,944	533

Table 3 - Performance comparison of translated and native images[66]

As can be seen by the sampling of benchmarks, performance varies over a significant range depending on the usage of the various libraries. While the best performance is undoubtedly achieved by fully native images, all performance tests are within an acceptable band. The use of shareable images does not make a significant difference as compared with the use of single images. Additionally, the identification of the actual source of high CPU utilization allows the conversion of that aspect of the program to natively compiled code on a priority basis, allowing engineering effort to be focused where it produces the largest payoff.

Summary

OpenVMS permits the staged conversion of production environments from the Alpha or VAX architectures to the Integrity platform without the need for interruptions, cold cutovers, or other high-risk strategies.

[63] Transfer of INFO-ZIP.ZIP (Distribution file for INFO-ZIP; 6,874 blocks)
[64] SPECint2000 result for rx 2620-2 (1.3GHz/2MB Itanium 2) from [SPEC2004]
[65] Transfer of INFO-ZIP.ZIP (Distribution file for INFO-ZIP; 6,874 blocks)
[66] For complete datasets from the test series see http://www.rlgsc.com/publications/vmstechjournal/migrationstrategies.htm

The features of OpenVMS allow the transition from older architectures to the Itanium architecture in a well-controlled fashion, consistent with technical requirements and corporate resources, without compromising ongoing 24 x 7 operations.

References

Bosworth, S., and Kabay, M., *Computer Security Handbook,* 4th Edition, John Wiley and Sons, 2002 [Bosworth2002].

"Business as Usual While Moving an Eight-Story Steel-Frame Building," *Engineering News-Record,* July 2, 1931.

Cocke, J., and Markstein, V., "The Evolution of RISC Technology at IBM," *IBM Journal of Research and Development,* Volume 34, Number 1 [Cocke1990].

da Cruz, F., and Gianone, C., *Using C-Kermit Communication Software, Second Edition*, Digital Press, 1997 [daCruz1997].

D'Antoni, G., "Porting OpenVMS Applications to Itanium," Compaq Enterprise Technology Symposium (CETS), 2001.

Dobberpubl, et al., "The MicroVAX 78032 Chip: A 32-bit Microprocessor," *Digital Technical Journal,* Volume 2, March 1986 [Dobberpubl1986].

Gezelter, R., "Alpha/Itanium Issues," July 2001; see `http://www.rlgsc.com/alphaitanium.html`.

Gezelter, R., "The Third Porting: Applying Past Lessons to the Alpha/Itanium Transition," Compaq Enterprise Technology Symposium (CETS), September 2001 [Gezelter2001].

Gezelter, R., "Inheritance Based Environments for OpenVMS Systems and OpenVMS Clusters," *OpenVMS Technical Journal,* Volume 2, February 2004 [Gezelter2004].

Gezelter, R., "Migrating OpenVMS Storage Environments without Interruption/Disruption," HP Enterprise Technology Forum, September 2005 [Gezelter2005].

Gezelter, R., "Code Portability and Related Issues for EPIC," IEEE Distinguished Visitor Vermont Speaking Tour, Dartmouth College, September 2005 [Gezelter2005a].

Gezelter, R., "Strategies for Enterprise Migration from Alpha and VAX to HP Integrity," Encompass Canada Local User Group and National Research Council (Canada), Institute for Information Technology, November 2006 [Gezelter2006].

Goldenberg, R., Kenah, L., *VAX/VMS Internals and Data Structures, Version 5.2*, Digital Press, 1991.

Goldenberg, R., Saravanan, S., *VMS for Alpha Platforms: Internals and Data Structures*, Preliminary Edition, Volume 3, Digital Press, 1993.

Goldenberg, R., Dumas, D., Saravanan, S., *OpenVMS Alpha Internals: Scheduling and Process Control,* Digital Press, 1997.

Gosling, J., Joy, B., Steele, G., *The Java Language Specification,* Addison-Wesley, 1996 [Gosling1996].

Grant, C., "Porting OpenVMS to HP Integrity Servers," *OpenVMS Technical Journal,* Volume 5, June 2005.

Gursha, J., *High Performance Cluster Configuration System Management,* Digital Press, 1997

IEEE Standards Committee 754, *IEEE Standard for Binary Floating Point Arithmetic, ANSI/IEEE Standard 754-1985,* IEEE, New York, 1985 (reprinted in SIGPLAN Notes, 22(2):9-25, 1987.

Intel® Itanium® Architecture Software Developer's Manual: Application Architecture," Revision 2.2, Volume 1, Intel Corporation, July 2006.

Intel® Itanium® Architecture Software Developer's Manual: Instruction Set Reference," Revision 2.2, Volume 3, Intel Corporation, July 2006.

Levine, D., *Software Development and Quality Assurance,* Chapter 25, [Bosworth2002].

Microcomputer Processors, Digital Equipment Corporation, 1978 [Digital1978c].

Reagan, J., "Porting the MACRO-32 Compiler to OpenVMS I64," *OpenVMS Technical Journal,* Volume 6, February 2005.

Rav, B., and Schlansker, M.S., "Explicitly Parallel Instruction Computing," *Computer,* February 2000 [Schlansker2000].
Rodin, G., "The 801 Minicomputer," *IBM Journal of Research and Development,* Volume 27, 1983 [Rodin1983].
Rusu, S., and Singer, G., "The First IA64 Microprocessor", *IEEE Journal of Solid State Circuits,* Volume 35, Number 11, November 2000.
Sites, R. (ed.), *Alpha Architecture Reference Manual,* Digital Press, 1992 [Sites1992].
Sites, R., "Binary Translation," *Digital Technical Journal,* Volume 4, Number 4, 1992 [Sites1992].
Strecker, W., "VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family," *Proceedings of the National Computer Conference, 1978* [Strecker1978].
TechWise Research, Inc., *Quantifying the Value of Availability: A Detailed Comparison of Four Different RISK-Based Cluster Solutions Designed to Provide High Availability,* Version 1.1a, June 2000 [TechWise2000].
TIS Committee, *Toolkit Interface Standards (TIS) Executable and Linking Format (ELF) Version 1.2*, May 1995; from http://www.x86.org/ftp/manuals/tools/elf.pdf, December 2006.
van der Burg, J., "Disk Partitioning on OpenVMS: LDDRIVER," *OpenVMS Technical Journal,* Volume 6, June 2005.
VAX-11 Architecture Handbook, Digital Equipment Corporation, 1978 [Digital1978].
VAX-11 Software Handbook, Digital Equipment Corporation,1978 [Digital1978b].
Zahir, R., et al., "OS and Compiler Considerations in the Design of the IA-64 Architecture," *Ninth International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS-IX),* Association for Computing Machinery.

OpenVMS manuals

HP OpenVMS DCL Dictionary: A–M (September 2003, Order number AA-PV5KJ-TK)
HP OpenVMS DCL Dictionary: N–Z (September 2003, Order number AA-PV5LJ-TK)
HP OpenVMS Systems Manager's Manual, Volume 1: Essentials (September 2003, Order number AA-PV5MH-TK)
HP OpenVMS Systems Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems (September 2003, Order number AA-PV5NH-TK)
OpenVMS Guide to System Security (June 2002, Order number AA-Q2HLF-TE)
OpenVMS User Manual (January 1999, Order number AA-PV5JD-TK)
OpenVMS Version 7.2 New Features Manual (January 1999, Order number AA-QSBFC-TE)
HP OpenVMS Programming Concepts Manual, Volume I (January 2005, Order number AA-RNSHD-TE)
HP OpenVMS Programming Concepts Manual, Volume II (January 2005, Order number AA-PV67H-TE)
HP OpenVMS LINKER Manual (July 2006, Order number BA554-90004)
HP OpenVMS Porting Applications from HP OpenVMS Alpha to HP OpenVMS Industry Standard 64 for Integrity Servers (January 2005, Order number BA-442-90001)
HP OpenVMS Migration Software for Alpha to Integrity Servers: Guide to Translating Images (February 2005)
HP OpenVMS Migration Software for Alpha to Integrity Servers: Release Notes (February 2005)
VAX MACRO and Instruction Set Reference Manual (April 2001, Order number AA-PS6GD-TE)

Further reading

Davis, R., *VAXcluster Principles*, Digital Press, 1993.

Acknowledgments

I would like to thank the numerous people who generously took the time to speak with me about the technologies involved in the architectural transitions in the OpenVMS community; the initial transition from the 16-bit PDP-11 RSX-family to VAX/VMS; the transition from the CISC VAX architecture to the RISC Alpha architecture; and the transition from the RISC Alpha to the EPIC Itanium architecture. I would like to thank members of OpenVMS Engineering, including Andrew Goldstein, Gaitan D'Antoni, and others who contributed recollections and pointers to information. I would also like to thank John Streiff, James Gursha, Jerrold Leichter, Gideon Eisenstadter, and others who read drafts and contributed their comments. I would also like to thank Fern Hertzberg for her assistance in organizing and copyediting this paper.

Biography

Robert Gezelter, CDP, CSA, CSE, Software Consultant, guest lecturer and technical facilitator has more than 29 years of international consulting experience in private and public sectors. He has worked with OpenVMS since the initial release of VMS in 1978, and with OpenVMS Cluster systems since their announcement in 1982. He has worked with portable software, translation, and cross compilers since 1975.

Mr. Gezelter received his BA and MS degrees in Computer Science from New York University. He also holds Hewlett-Packard's CSA and CSE accreditations relating to OpenVMS.
Mr. Gezelter is a regular guest speaker at technical conferences worldwide such as the HP Technology Forum and Encompass (formerly DECUS) events. His articles have appeared in the Network World, Open Systems Today, Digital Systems Journal, Digital News, and Hardcopy. He is also a Contributing Editor to the Computer Security Handbook, 4th Edition (Wiley, 2002) and the author of two chapters in the Handbook of Information Security (Wiley, 2005), including the chapter on *OpenVMS Security*. Many of his publications and speeches are available through his firm's www site at http://www.rlgsc.com.

He is a Senior Member of the IEEE, and a member of Infragard, Encompass, and International Association of Software Architects. He is an alumnus of the IEEE Computer Society's Distinguished Visitors Program and is a Director of the New York City chapter of IASA.
His firm's consulting practice emphasizes in-depth technical expertise in computer architectures, operating systems, networks, security, APIs, and related matters.
His clients range from the Fortune 10 to small businesses, locally, nationally, and internationally on matters spanning the range from individual questions to major projects.
He can be reached via his firm's www site at http://www.rlgsc.com.

OpenVMS: Striving to provide the support you need

Ted Saul, OpenVMS Offsite Consultant and Global Project Manager

Introduction

It is no secret that support for HP operating systems and accompanying software is changing. New phone numbers, interactive voice response systems (IVRS), and unfamiliar voices on the other end of the phone all present customers with new challenges and frustrations. OpenVMS is not immune from HP's outsourcing strategy, and much discussion has taken place regarding the issues of the program. There is no denying that, during the initial rollout to outsource front-line support, problems surfaced and customer satisfaction suffered. This transition has also proved to be challenging for OpenVMS staff who have been supporting and investing their energies in the product for years. These are also the same people that you have worked with many times. As the HP advocates closest to the customer, we felt it was important to involve ourselves with the areas that we could influence. Presented with the new direction of the company, we needed to the identify gaps in support and to develop plans to improve those areas. The goal of this article is to give a behind-the-scenes view of to the actions we have taken to consistently strengthen the support of the OpenVMS product while adhering to HP's outsourcing strategy.

It became clear that training and retraining would be the key to a successful support program. Even though the initial support team spent hours in the classroom, we knew that a process to strengthen learned skills was required. The call center identified areas of OpenVMS functionality that generated the highest volume of calls, and then targeted those areas for developing training. Key to the success of the training would be development of the curriculum by call center engineers who had the most experience working with customers and solving problems. This would ensure that the proper topics were covered to assist with identifying and troubleshooting problems as well as knowing the appropriate questions to ask.

Training was delivered using two methods. First, call center personnel traveled to the HP site in Bangalore to present the training in person. Not only did this provide the best learning experience, but it also developed relationships among staff. In addition, in-person training helped employees to understand the environment their counterparts deal with in their daily jobs. Bangalore staff could ask questions easily and get answers clearly. Second, remote training was held via conference calls and virtual classrooms. This allowed for the involvement of many experience OpenVMS engineers as well as a focused road map of training. Remote sessions allow for standard instructor-led training as well as ongoing seminars and brown-bag programs to ensure that new engineers are kept current about many different OpenVMS products.

In addition to the constant schedule of training, new engineers are assigned mentors who are seasoned OpenVMS professionals. Engineers are paired according to time shifts so that questions can be addressed in real time and direction can be given as necessary. Mentors can also help identify knowledge gaps and suggest the appropriate training. Along with one-on-one mentoring, chat tools are available to internal support staff that allow for immediate access to additional resources. These chats are monitored 24 x 7 x 365 by OpenVMS experts from around the world. Participants include not only support engineers, but Ambassadors, Proactive services representatives, and managers in the OpenVMS community, all of whom share the goal of continuing the success of the OpenVMS operating system. Once an individual raises a question regarding an OpenVMS issue, suggestions, solutions, and even offers to participate in solving the case are generously received. This type of collaboration can help new engineers grow and develop the same levels of expertise as engineers who have many years of experience.

Along with specific training and mentoring, we have assigned individuals to projects that are focused solely on resolving issues related to outsourcing. The individuals are very familiar with the new teams, management, and structure, and handle communications that aim to improve efficiency and

performance. If an issue needs to be addressed, these project leaders act as intermediary, ensuring that correct information is clearly communicated to the teams or individuals involved. This approach ensures the availability of resources that are required for providing elevated support.

Measurements and ratings have also been developed to make sure that training and mentoring goals are realized. Although these measurements are not new to customer support, they have taken on an even more important role with the new strategy. One method is the customer survey. Each customer survey response is taken very seriously, and every concern is addressed appropriately. In addition, a separate team is dedicated to listening to incoming calls to observe English skills and other cultural differences that might need attention. Each call is recorded, and questions about its handling can then be reviewed by managers and other HP personnel.

Probably the most exciting initiative taking place in OpenVMS is the globalization of customer support. HP has always been a global company, but efforts in services to join its global resources together to work as one entity has been elevated to higher priority in the last few years. OpenVMS is working hard to lead the way in this endeavor. The global chat channel mentioned previously is just the beginning. Organizational change opportunities are being taken advantage of to bring resources with like skills together no matter where they are located in the world. As of late last year, OpenVMS experts in HP's Australia, Pacific, and Japan (APJ) region began reporting to a manager from the America's region. Additionally OpenVMS teams in Europe work closely with the America and APJ team to provide 24x7x365, seamless support. Processes are in place to pass calls between the regions when necessary. An example of successful global support occurred recently, when a customer in the Americas Region needed an expert in cluster configuration to be standing by in the early evening. Rather than require the customer to pay for an off-duty engineer, agreements were made for an APJ resource to be available at the appropriate time. Although the resource was required at night in the United States, the APJ engineer could be engaged during their mid-morning hours. The flexibility of a large global team also allows regions to cover for each other without hesitation during holidays and paid-leave periods. Although globalization presents its own challenges, these are easily outweighed by its achievements in providing efficient support. How does the globalization of OpenVMS support help customers? It ensures that an expert can be engaged any time of the day, anywhere in the world. Most important, these resources are available to the initial problem response representative.

We always welcome feedback and suggestions from our customer base. In order to keep the reputation of OpenVMS and its support mechanism at its well-known levels of quality, we need input from all involved. We encourage customers to share their good and bad experiences with us, along with ideas for improving their support experience. Although we are working within the strategy set by HP management, our goal continues to be providing high levels of quality of OpenVMS support under any constraint we face.

For more information

Questions, feedback, and suggestions regarding this article can be directed to , , GCC Manager or to any HP Services representative.

Java and OpenVMS: Myths and realities

Jean-Yves Bourlès and Thierry Uso, Consultants

Overview

This article discusses certain myths regarding the use of Java™ programs in the OpenVMS environment. We have identified three myths that, although dispelled by the facts, remain prominent within the OpenVMS and Java communities:

- Java is slow.
- Java is poorly adapted to OpenVMS.
- Java is totally portable.

Myth 1: Java is slow.

The presumed performance problems of Java programs are often used as an excuse for not deploying Java applications in the OpenVMS environment.

Java is neither a compiled nor an interpreted language, in the usual sense of the terms. Rather, Java is a virtual machine (VM) based language. The compilation of Java source code results in a bytecode per Java class, and the bytecodes are platform independent (processor type and OS). The VM interprets the bytecodes by translating them on demand into machine instructions. The principle of the VM means that a Java-written program generally runs slower than one written in a compiled language (C, Ada, and so on) but runs faster than one written in interpreted language (PHP, Perl, and so on). Confirmation of this can be found in the microbenchmark results published on the Computer Language Shootout Benchmarks website (see Figures 1 and 2). The benchmark environment was AMD Sempron, Debian unstable Kernel 2.6.8-1-k7, Java HotSpot 1.4.2_05-b04, C gcc 4.0.3, PHP 5.1.2-1. However, we obtained similar results with Intel® Itanium® rx1620, OpenVMS Version 8.3, Java HotSpot 1.5-2, HP C 7.1-11.

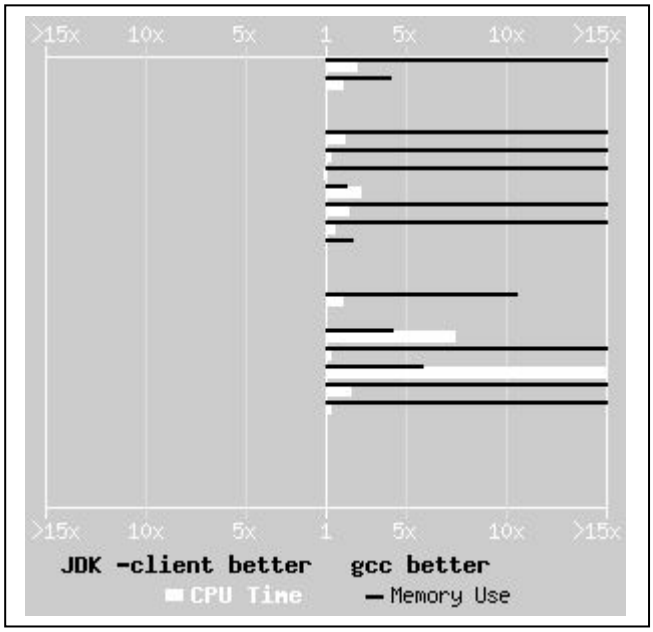


Figure 1: Performance of Java program compared with C program

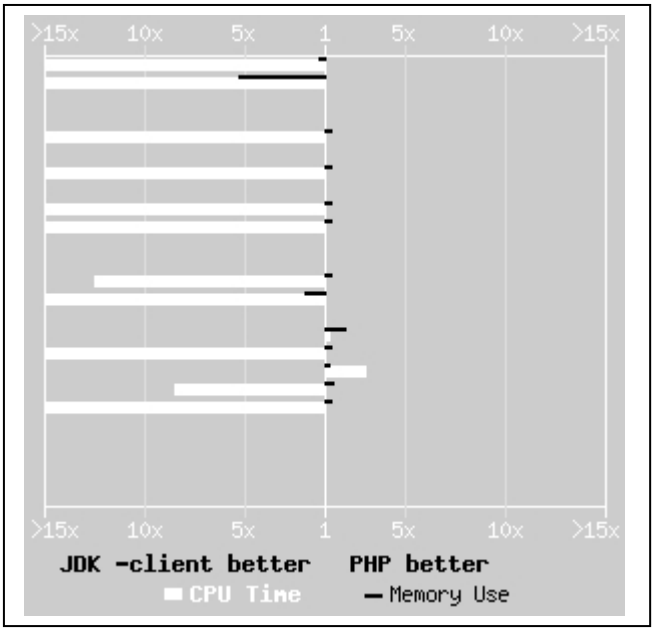


Figure 2: Performance of Java program compared with PHP program

These results show that the progress made in the conception of Java VMs through the introduction of the JIT (Just in Time) technology greatly reduces the performance divide between Java and compiled languages. On the whole, the quality of the written code has a greater impact on performance than does the choice of Java or a compiled language.

Failure to deploy Java applications in an OpenVMS environment can sometimes be justified by the VM-related increased memory consumption, but is increasingly less justifiable for the reason that the language is "slow."

Myth 2 : Java is poorly adapted to OpenVMS.

Launching certain Java applications is very slow on OpenVMS systems. For example, launching the Jetty application server (with the deployment of the web application `Javadoc` from the javadoc.war archive) on a DS20 2x500MHz with 1GB of RAM takes 105 seconds on OpenVMS Version 7.3-2 (Fast VM 1.4.2-3), as compared with 11 seconds on Tru64 UNIX® Version 5.1B (Fast VM 1.4.2-4).

Exactly how poorly adapted to OpenVMS is Java? To answer this question and to identify the reasons for this slowness, we carried out several benchmarks on the platform described in the preceding paragraph.

Benchmark 1 was based on a program that calculated binary search trees; the program is an adapted version of the `Binarytrees` program found on the Computer Language Shootout Benchmarks website. For a depth of 18, the results were as follows:

Operating System	Delay (seconds)
OpenVMS	37.4
Tru64 UNIX	27.7

Benchmark 2 was based on a program that sequentially read a file, byte by byte, using the `InputStream()` function. For files of size 10, 100, and 1000 KB, the results were as follows:

Operating System	Delay 10KB (seconds)	Delay 100KB (seconds)	Delay 1000KB (seconds)
OpenVMS	5.0	34.1	332.7
Tru64 UNIX	8.0	79.0	808.4

Benchmark 3 was based on a program that sequentially read a file, block by block, using the `BufferedInputStream()` function. For files of size 10KB, 100KB, and 1000KB, the results were as follows:

Operating System	Delay 10KB (seconds)	Delay 100KB (seconds)	Delay 1000KB (seconds)
OpenVMS	3.2	14.3	128.4
Tru64 UNIX	0.1	0.7	6.9

Only the third benchmark shows significant differences between OpenVMS and Tru64 UNIX. In reality, the poor performance of the buffered I/O of OpenVMS compared with Tru64 UNIX, can be explained by the slowness of the file system (RMS + ODS-5) and not by a bad implementation of the `BufferedInputStream()` function. In fact, the same order of difference is seen with the C program `unzip`.

In the case of the launch of Jetty under OpenVMS, the `Javadoc` deployment requires the most file system intensive use and is unsurprisingly the most costly. Of the 105 seconds taken for the launch, 96 seconds were used for decompressing the `javadoc.war` archive into 42 folders and 549 files.

Solutions exist to compensate for the file system slowness of OpenVMS, although no single solution alone suits all Java applications. In the case of Jetty, by default this application server deploys web applications (war archives) into a temporary folder where it also stores the bytecode of the servlets that are created in real time. The redirection of this temporary folder toward a RAMdisk decreases the launch time from 105 to 18 seconds and noticeably improves the response times of the web applications. Another solution that allows noticeable gains is to use only web applications that are deployed manually and with precompiled JPSs.

To summarize, Java is not poorly adapted to the OpenVMS environment. In reality, file system intensive applications are always slower on OpenVMS than on UNIX or Windows® systems, regardless of whether the applications were developed in Java. The only criticism that we can make of Java is that it frequently creates large numbers of application files (a bytecode per Java class).

Myth 3: Java is totally portable.

The creators of Java conceived an environment (VM language and standard libraries) that allows development of programs with maximum portability. In theory, it is the VM, not the developer, that deals with all the OS and hardware-related specificities.

Can we say, then, that Java is totally portable, as Sun implies with its slogan, “Compile once, run everywhere”?

The numerous tests and portings that we have performed in the OpenVMS environment show that Java programs are much more portable than those written in a compiled language. However, we are still a long way from verifying total portability. The reason is simple. The Java library creators, VM developers, and application developers are mainly from the UNIX world and, therefore, are unfamiliar with OpenVMS. (The OpenVMS VMs, Fast VM for Alpha, and HotSpot VM for Itanium systems, have been ported from Tru64 UNIX and HP-UX, respectively; the majority of Java applications were developed on and for UNIX platforms.)

Portability problems can be divided into two categories:

- Problems for the OpenVMS system engineer
- Problems for the Java application developer

The problems the system engineer encounters most often concern files and DCL, and these problems can be resolved without recompiling the application. For a description, see the *User Guide for Java for OpenVMS.*

Java deals only with files whose format is Stream-LF. Consequently, the system engineer must ensure that all the Java application files (bytecode, `jar` archive, data files, and so on) respect this format. Having originated in UNIX, the majority of Java applications use file names that do not respect ODS-2 file semantics. If this is the case then it is vital that the file system be migrated to ODS-5 format immediately, and that the Java application process is run with the correct parameters. For example, the following commands are often used:

```
$ SET PROCESS/PARSE=EXTENDED
$ DEFINE DECC$EFS_CASE_PRESERVE ENABLE
$ DEFINE DECC$ARGV_PARSE_STYLE ENABLE
$ DEFINE DECC$EFS_CHARSET ENABLE
$ DEFINE DECC$EFS_CASE_SPECIAL ENABLE
$ DEFINE DECC$ENABLE_GETENV_CACHE ENABLE
$ DEFINE DECC$POSIX-SEEK_STREAM_FILE ENABLE
$ DEFINE DECC$FILE_SHARING ENABLE
$ DEFINE JAVA$FILENAME_CONTROLS 8
```

In the preceding list, the first two commands force the process to respect case, and the last command defines the UNIX and OpenVMS file-name mapping rules that the VM must use. Sometimes choosing the good value for these logical names can be a delicate matter.

Java ignores the notion of file versions. Because of this, a good practice is to limit file versions to one on all files in the Java application file hierarchy. To do this, use the following commands on the root folder of the hierarchy:

```
$ SET DIRECTORY/VERSION LIMIT=1 disk:[rootfolder...]
$ SET FILE/VERSION_LIMIT=1 disk:[rootfolder...]*.*;*
```

The remaining task is to translate the application startup scripts into DCL procedures. You must use the foreign command `JAVA` to launch Java applications. This command can take either of the following two forms:

```
$ JAVA bytecode
$ JAVA -jar archive-jar
```

The bytecode name corresponds to the Java class name in the source code. Because Java is case sensitive (unlike DCL) the bytecode name must be quoted to deal with this difference. For example:

```
$ JAVA "MyClass"
```

The `JAVA` command accepts several optional parameters, including the `classpath`, which indicates the location of the bytecodes. These parameters can sometimes cause the final command

line to exceed the DCL limit for commands within a procedure. For OpenVMS Version 7.3-2, this limit is set to 8192 bytes, and the `JAVA` command must be modified to respect this limit. OpenVMS provides various workarounds to accomplish this (for example, `option` file, `classpath` file; logical names such as `JAVA$ENABLE_ENVIRONMENT_EXPANSION`, `JAVA$CLASSPATH`, and so on).

The problems the Java developer encounters most often concern process management (such as fork, environment variables) and can be only resolved by modifying the source code and recompiling the application. These problems are illustrated by the iReport application, a reporting tool drawn from Microsoft® Access and Crystal Report.

The iReport tool uses an external program (for example, XPDF or Mozilla) to display the report in PDF or HTML format. This program runs within a detached process. On OpenVMS, the display works fine locally but fails remotely. The reason is that the X11 parameters (`display node`, `display transport`) are logical names that are not passed to the child process (display) by the parent process (iReport). We had to modify the source file (`iReportCompiler.java`) in order for the iReport process to pass the X11 parameters to the process controlling the display.

Use of the logical name `JAVA$EXEC_TRACE` can be useful for diagnosing portability problems related to process management. Once set to "true" in the LNM$JOB table, this logical name forces the VM to trace calls to the POSIX function `execv()` as well as its list of arguments.

Acknowledgments

The authors thank Tim Oakley for his French-to-English translation of this paper.

Trademarks

For more information

Useful links:

The Computer Language Shootout Benchmarks website: http://shootout.alioth.debian.org/
Documentation of Java on OpenVMS: http://h18012.www1.hp.com/java/documentation/
Jetty and iReport for OpenVMS: http://vmsfree.free.fr/freen/index.php

Contacts:
jeanyves.bourles@free.fr
t.uso@laposte.net

Implementation of a web application maintenance and testing environment

Willem Grooters, VX Company BV, OpenVMS developer and system manager

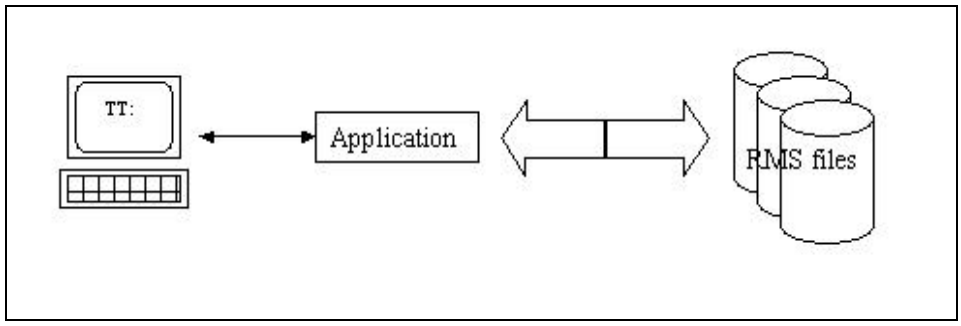
Introduction

Most locations where an application is developed and maintained have some kind of separation between the environments of code storage, actual development (the programmer) and testing. In classic environments, this is usually sufficient. Development and testing of web-based applications is more complicated because each environment requires a web server to debug and test the application. It gets even more complicated if more than one web server must be supported. This article describes how such an environment can be configured.

The application and its history

The basis for this article is an application, used by law enforcement, written in DIBOL in the 1980s, originally for VAX systems accessed on VT terminals. In the mid-1990s, the application was ported to the Alpha architecture.

The architecture of this application has been straightforward from the beginning. Data is stored in indexed files, most of them having multiple keys. The application includes just a few main images for data entry (which can be quite complex in structure and coherence) and data maintenance, and a few executables for application management:



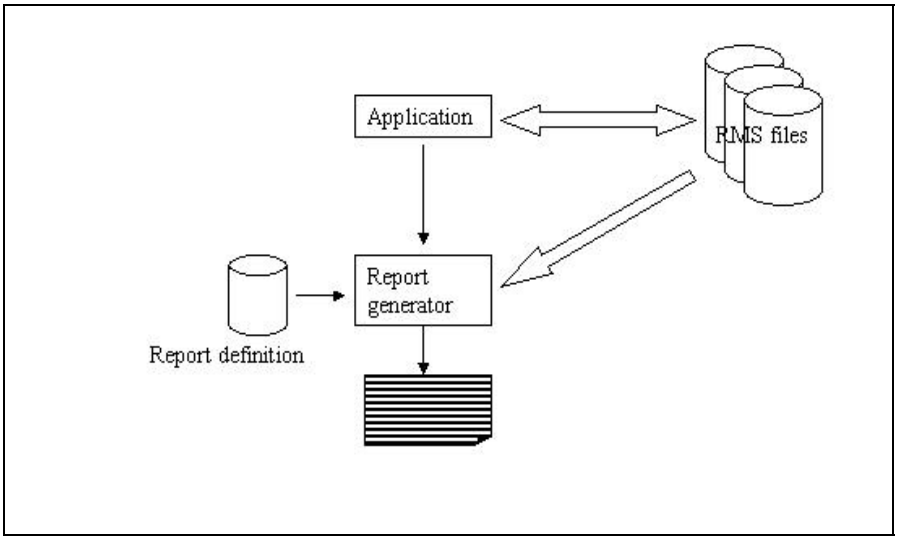
The application is maintained centrally but is used in over 20 different, largely independent locations – with their own system and application management.

Each location has its own group of users. These are end users who access the application programs by a menu system (either built in or locally maintained DCL procedures), allowing them to access application programs based on their functional profile, which is stored and maintained as part of the application. Some have access to just the basic functionality, others can access higher privileged functions, and a few -- the application managers – can access the most privileged functions and programs for maintaining data consistency, adjustment of application parameters, and a number of system management tasks.

As with any application, reports are created, some as a result of the normal functionality, some at the request of the user or application managers. Most of these reports are required for further legal activity, and some hold management information. Some are coded within the application, but localized reports were a requirement. Because of the nature of the application, some reports are subject to change as a result of government legislation – and sometimes quite drastically. Having these reports coded within the application meant a huge amount of work. For the local reports, the required customizations would have caused maintenance to become error prone or impossible.

For that reason, a report generator was introduced that is started by the application with some controlling data, to produce a report from the RMS files. The definition of the report includes layout, selections, filters, and transformations, and is coded and stored separately. These features make management of the reports easier.

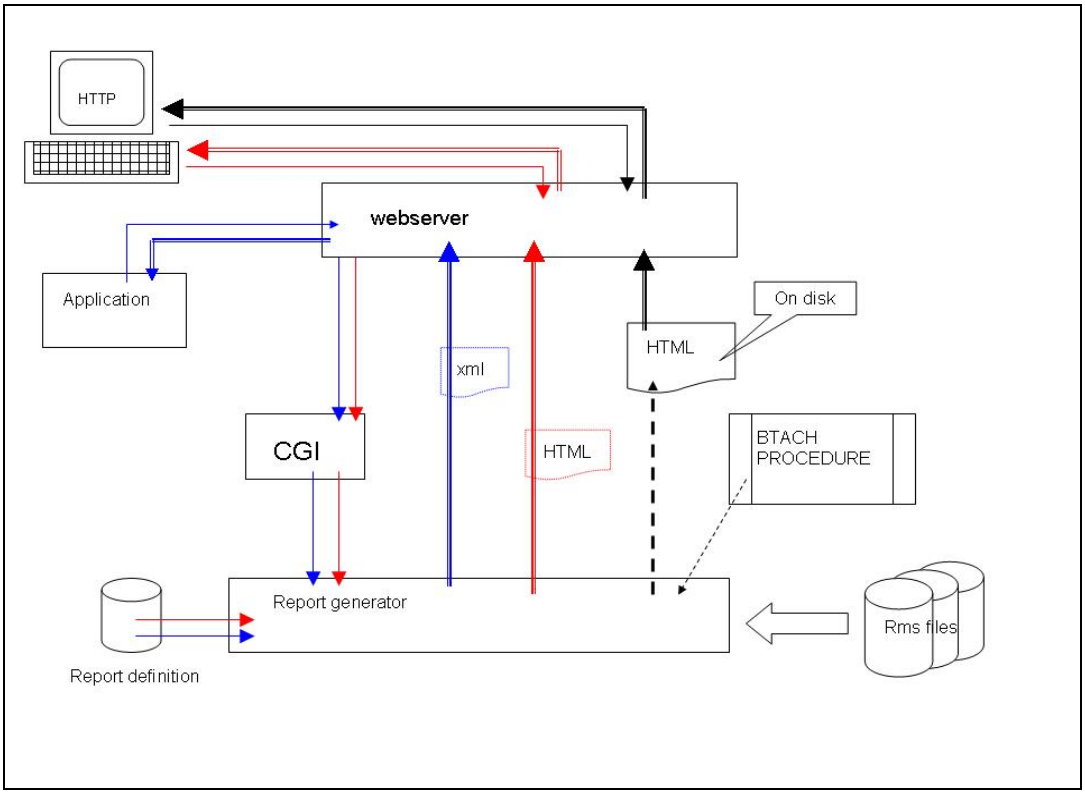
Reports can be created by the application, either as part of the workflow or in batch mode. The following figure shows the flow of data to and from the report generator.



Users of this application also needed the ability to create and modify reports interactively -- if only to test the application. Reports were developed that could be created without intervention of the application itself. Report generation could be started by a terminal session, but a batch procedure was also a possibility.

The mid-1990s saw a drive for accessing the data in a web browser. Because reports could be created in batch, it was a small step to develop report definitions that created HTML pages that were accessible by a web browser over a web server.

In the beginning, these reports were created daily in batch mode. Later, interactive access was introduced: static pages allow the user to enter search criteria, thereby launching a CGI procedure that starts the report generator. The report generator then creates pages that, in turn, can create lists and data reports and can even display reports that have been created from the application. The following figure shows this process:



Legend:
Black: access of report, generated in batch and stiored on disk
Red: access of data based on selection specified in URL,, returns data in HTML (or xhtml) format
Blue: Access of data based on selection specified in URL, returns data in XML format.
Single line: request
Double line: returned output

The original application, however, remains the only way to add and modify data and to do the main processing on the system. The web interface is used only as another means to view the data. The only exception is the possibility of uploading files that are related to existing data, but these uploaded files can be accessed only using the web interface. The data that constitutes the link, however, can be accessed and modified (by authorized personnel only) by a normal executable.

A word about the report generator

The basis of reports are report definitions, that is, plain text files containing simply a piece of code that describes what is being read from the application files, based on data that is passed by symbols or logicals, on data that is stored within the data files, or that is hardcoded in the definition. Specifically, the code describes what filters and conversions are applied to the read data and how the information is displayed on SYS$OUTPUT – to the terminal, in a file to be stored on disk or sent to a printer, or, for web requests, input to the web server that sends the output to the requesting browser.

This code is compiled into a byte-code file using indexed files that describe the application's files in terms of records, fields (their offsets, data type, and size), and keys (key number, starting position, and size). This byte code is input to the report generator to create the report.

The report generator is originally developed as part of the application, but in such a way it can be used by other applications as well. The files describing the files in the application, the report descriptions, and the resulting byte code, however, are part of the application.

Versioning

All elements of this structure -- the application and the web interface -- are closely related. The report generator is an application by itself, but it processes report definitions that are part of the application or web interface. To process these report definitions, the generator needs data about the application files to create the output. Such data includes keys, location of data files within the records, and naming of these fields, and so on. These files are specific to the application. When a file changes, as with the removal of a key or the addition of a field in the record, the files need to be updated as well.

By these generator files that describe the files, the report generator can now exactly locate the fields from a file. If any of these fields moves, the wrong data is shown. Therefore, the right files need to be addressed.

Processing the report definitions actually means compiling them into byte code to be interpreted and executed by the generator. The compiled version is specific to a given version of the application or web interface. However, in some cases, changes to the report compiler or generator require recompilation of the report definitions. For that reason, the release of a new report generator always coincides with a new release of the application and web interface. Therefore, for the web server, it is important to access the right version of both the web interface and report generator.

Application structure

It is important to realize that the development team has no control over the versions used in each of the locations. Two or three versions might be active and each needs to be maintained. Also, moving from one version to another can mean that two versions are temporarily accessible at the same time at one location, and both versions must be maintained simultaneously. Therefore, for any given activity, the user must always be aware of what version is in use.

At some locations, different departments use different data sets. That is, the files are identical in structure and name, but their content is different. Within the location these are called "administrations"; externally, the only difference is the number used in the physical name. For instance, one department can have number 001, the other 003. Thus, a file logically named HFD is named HFD001 for the first department, and a similar file for the other department is named HFD003. (This is part of the DIBOL structure used in the original environment.) The structure within the application is quite strict and is the same across all versions, all data, and all administrations.

The application root contains all versions of the application that are kept on line, and all of them have the same structure, as shown in the following example:

```
ROOT
   2002A
      ADM BIB BOB COM EXE FDL GEG LIB LST MNU REC REF RPT SRC
   2002B
      ADM BIB BOB COM EXE FDL GEG LIB LST MNU REC REF RPT SRC
   2004A
      ADM BIB BOB COM EXE FDL GEG LIB LST MNU REC REF RPT SRC
   2006A
      ADM BIB BOB COM EXE FDL GEG LIB LST MNU REC REF RPT SRC
```

The report generator has a similar structure.

This structure is used for all working environments, from programming to production, where the production environment would miss some of them – obviously the ones containing source code. Each of these directories is accessible using a single logical name that is set by a command procedure and is maintained within each version.

The web application has a similar structure but with one addition: the entry point for the web server is shared by all versions, and the web server has no data about application versions. The following example shows this structure:

```
CGI
   APPLWEB
      COM
   APPLROOT
      2002A
            BOB COM EXE REC RPT SRC
      2002B
         BOB COM EXE REC RPT SRC
      2004A
         BOB COM EXE REC RPT SRC
      2006A
         BOB COM EXE REC RPT SRC
```

Physically, the version structure of the web application is stored within the application structure and is separately rooted, as shown in the following example:

```
ROOT
   2002A
      ADM BIB BOB COM EXE FDL GEG LIB LST MNU REC REF RPT SRC WEB
                                                    BOB COM EXE REC RPT
SRC
   2002B
      ADM BIB BOB COM EXE FDL GEG LIB LST MNU REC REF RPT SRC WEB
                                                    BOB COM EXE REC RPT
SRC
   200$A
      ADM BIB BOB COM EXE FDL GEG LIB LST MNU REC REF RPT SRC WEB
                                                    BOB COM EXE REC RPT
SRC
```

The CGI directory, however, is kept outside the structure because it is part of the web server.

The production environment matches this structure based on logical names. Most users have these logicals defined in their login command files or have them defined by a menu system.

The web interface, however, is not part of the application in the production environment; rather, it is a separately installed product. The CGI procedure that is started by the web server determines a name based on application, version, administration, and server; looks for a logical with that name; and runs the corresponding command procedure. That procedure then sets all the required logical names that refer to this web interface and to the application data.

Development environment

Development and maintenance of the application is structured into a number of working environments.
The central, common environment is Development (DVLP). This is used as a container for files that are used to create a new release. There is no direct development in this environment.

Each programmer has a separate working area, following the structure described in the preceding section and referred to by the same logicals as any would be referred to, but now set up as a search list containing the programmer's own environment first, followed by the DVLP environment.

Although the DVLP environment contains the web interface as part of the application, this is not true for the programmer's environment. In the latter, the application and the web interface to it are two distinct applications, even though they share the same data files. The primary reason for this distinction is to prevent unintended interaction between the two.

For the application itself, testing in the programmer's environment does not differ from running the application in a normal way. However, the web interface cannot be tested that way. This is one of the problems to solve.

Different web servers

Various web servers are available on OpenVMS, and any of these can be used in a production environment.

The first web server that is used in all locations is the Ohio State University web server, known as the OSU web server. At the time, this was the only web server that was free and that offered sufficient capabilities and secure access. This web server relies on DECnet for its CGI scripting, and it requires the following specific commands in the CGI command procedure for the processing:

```
$ write net_link "<DNETRECMODE>"        ! Set implied carriage control.
$ mcr www_root:[bin]CGI_SYMBOLS "WWW_" "FORM_"
$!
$! Send back CGI response.  Note that newlines (<CR><LF>) must be
$! explicitly sent.
$!
```

Since DECnet was to be phased out after 2000, there is the need to switch web server – and Secure Web Server (SWS) was introduced. This is based on the well-known UNIX-based Apache server. In testing the application with this server, it was soon discovered that the CGI procedure needed to be changed (for example, all OSU-specific code needed to be removed), and that the server handled spaces and other characters very differently. Consequently, the report definitions of the web interface needed to be changed as well. Since not all locations could transition immediately, the CGI procedure for SWS was different from the one from OSU.

The reports needed to change as well. The OSU web server has no trouble with characters like space, greater than and less than, colon, and so on. However, SWS cannot handle these characters properly, so translations needed to be done. To avoid having separate code for OSU and SWS, translation requirements were identified by checking for the logical APACHE$COMMON, and if that existed, translations were executed as necessary. The end result was one code base for the report definitions but two distinct kits for the web interface to be released: one for OSU and one for SWS.

New developments were now first tested on SWS, but testing them on OSU was troublesome and soon showed that, in some cases, drastic changes were required for the new development to be usable on the OSU web server. This situation could have been prevented if testing under this server had been possible from the beginning.

Later it was discovered that SWS was not able to handle mixed-case passwords, a requirement in some locations. The WASD web server was introduced as an alternative for this reason. Research revealed that, except for a few differences, the CGI procedure as defined for SWS was usable.

At that time, a complete overhaul of the web interface was started in one programmer's environment, whereas others started new development and did regular maintenance. These activities required the configuration of a web server environment that facilitated access to different locations without interfering with access to others. It also meant that not just one web server was available, but all that applied to the organization were available.

The challenge

The issue now was how to create an environment where:

- Each programmer has a "web" to develop and test as follows:
    - Without interfering with other programmer's work
    - In a near-production structure
    - By each of the three web servers concurrently
- The version that is delivered can be accessed as follows:
    - By each web server
    - In a standard structure

The second point (versioning) was particularly important because, at that moment, no such environment existed.

The implementation

Installing different web servers is not a big problem because their root directories are different, as are their logical names. The main issue is their configuration – each should have its own set of ports to listen to.

The "reference" system – that is, the system that has been delivered – gets a port on each server. Since each programmer will use each of the three web servers, each programmer was assigned a port on each server, to be used in the URL to access the specific environment.

Added to these was the need for a port to access the web interface in the DVLP environment, and one for a test environment (the version to be delivered) to allow regression and acceptance testing on each server.

Actually, this is what "multiple webs" is all about: one web server serving a number of different, unrelated webs. All three servers support multiple webs, even though their naming differs.

It was decided that one machine – separate from DVLP and, therefore, from the programmers – would be set up for testing and research.

On the DVLP system, configuration for each web server was as follows:

- A virtual web for the reference system

- A virtual web for the DVLP system
- A virtual web for each of the developers

On the test system, the configuration contained:

- A virtual web for testing HTML interface
- A virtual web for testing application access
- At least one virtual web for research

Because of development of a common CGI procedure, and the requirement to start with renewal of the reports, we started with the development machine. Here, only SWS 1.3 was installed, and testing on the OSU web server could be done on another machine.

For maintenance reasons, the web servers were installed off the system disk under a single root directory, as shown in the following example:

```
APPL_DISK
   WEBSERVER
      OSU            ← WEB$ROOT
         OSU structure
      APACHE      ← APACHE$ROOT
            SWS structure
      HT_ROOT     ← HT_ROOT:
```

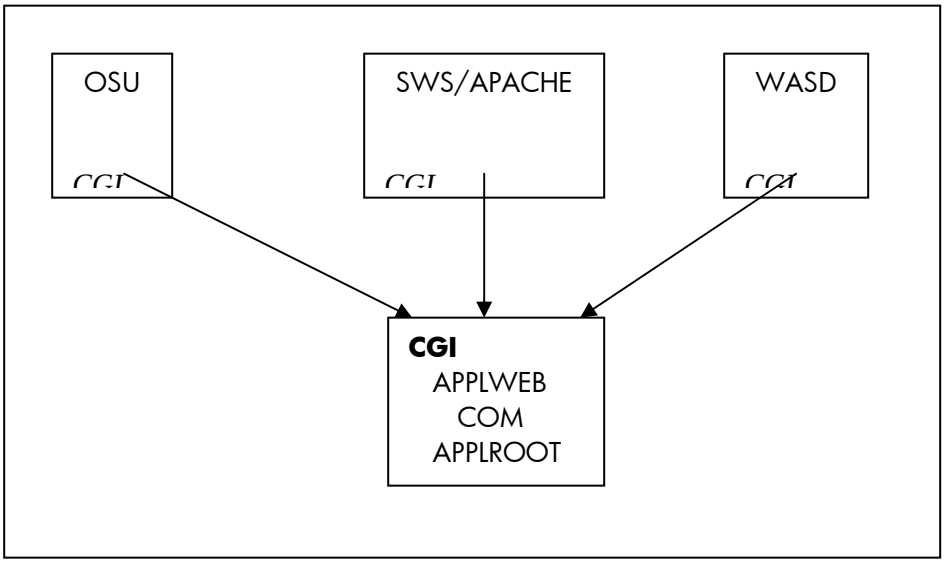
The next point was to map the structures for the web server in such a way that it would be identical for each of them: Each web server needed to see the identical structure. That meant only a single CGI procedure could handle all requests. Another requirement was that the URL used could not contain any data that was specific for one server, implying that any differences would need to be handled by the CGI procedure. Furthermore, changing the URL was out of the question.

The mapping of the structure meant correct definition of logical names. This was already present in both the OSU and SWS web server environments in the production environment, and so needed to be set up in the DVLP and the programmers' environments. Because a single command procedure was to handle all three servers, this issue was moved to the development of that procedure.

First the structure that is immediately accessed by the web server (that is, the CGI directory and the content below it) was to be set up. Because the access of the web interface requires authentication on first access, the standard [CGI-BIN] directory cannot be used, in stead CGI is a different directory outside the server structure: it is located in the environment it belongs to. The same applies to the web-interface files – however, these are physically located in the application environment as described earlier. The logical names however give it a look as a different application, as if it was a production environment.

This works fine for one version, but if different versions were to be accessed, a different CGI directory tree would be required for each version of the application. This does not conform to the production environment – one CGI procedure would have to be used for all versions. (The URL contains all the needed data.) Changing the configuration is not an acceptable solution because the process is error prone, and because it might disrupt existing connections by requiring the server to be restarted.

The solution was found by creating an alias for the directory into each of the web server roots (SET FILE /ENTER). That way, only one physical CGI directory tree is used by all three the web servers, as shown in the following figure:



Any changes anywhere in this directory tree can be directly accessed by any web server, so the whole environment can be tested in parallel. Access to this central location has become part of the procedures used by the deveolpment team.

On this system, just one environment was foreseen initially, so this structure was examined in that environment first. Each web server served on its own port: OSU on port 80 (because that was found to be the hardest to configure), SWS on port 82, and WASD on port 84.

We installed a very simple web application on the CGI directory – just a few HTML files and one procedure – as a proof of concept to show this method would work. After that, we started converting the existing SWS configuration on the development machine. The version that was last delivered was already installed on port 80, and we kept it there. Its configuration, however, was moved to the first "Virtual Host" entry in the configuration, as shown in the following example:

```
### Section 3: Virtual Hosts
## defined ports
## ------------------
# NameVirtualhost aaa.bbb.ccc.ddd
include conf/virthost.conf

## DEFAULT (port 80)
<VirtualHost _default_ aaa.bbb.ccc.ddd>
    ServerAdmin xxxxxx
    Documentroot "/apache$common/cgi
    ErrorLog logs/error_log
    CustomLog logs/access_log common
<Directory "/apache$common/cgi">
    Options Indexes FollowSymLinks Multiviews
    AllowOverride All
    Order allow,deny
    Allow from all
    AuthType Basic
      Authname "Live system"
    AuthUserOpenVMS on
    AuthAuthoritative On
    require valid-user
</Directory>
</VirtualHost>
```

Ports 81 to 90 were also mentioned as ports to listen to, and for each port a configuration file was created to be included in the server's configuration file. These files would hold the information of the programmer's location, so each programmer had a separate web.

Each programmer was to have a separate CGI directory tree that would resemble the live version I structure and files – including the CGI procedure used. Starting with the current SWS-related procedure, only one change was needed: where the live CGI procedure creates a logical to access the local definition of working environment, the setup for the programmer's environment could be served by a fixed one. Otherwise, a number of logicals would have to be defined, one for each programmer and this is not needed because the structure is well defined in this environment. This single procedure actually mimics the user's login when setting up the programmer's development area, causing the application logicals to be defined like the programmer would have them defined. That is, the logicals refer first to the programmer's directory, and second, to the DVLP area. Any procedure or image running would now be run as if the programmer started it, as shown in the following example:

```
## Port 86
##
## (WILLEM)
##
<VirtualHost aaa.bbb.ccc.ddd: 86>
    DocumentRoot /user/willem/cgi
    ErrorLog logs/willem-error_log
    CustomLog logs/willem-access_log common

<Directory "/user/willem/cgi">
    Options Indexes FollowSymLinks Multiviews
    AllowOverride All
    Order allow,deny
    Allow from all
    AuthType Basic
    Authname "WILLEM"
    AuthUserOpenVMS on
    AuthAuthoritative On
    require valid-user
</Directory>

# Scripting is version independent
ScriptAlias /cgi/ "/user/willem/cgi/appl/com/"

# Application web interface files version 2004
Alias /appl/04/WebIF/ "/user/willem/WebIF04/"
Alias /appl/04/ "/user/willem/WebIF04/"

# Application web interface files version 2004
Alias /appl/06/WebIF/ "/user/willem/WebIF06/"
Alias /appl/06/ "/user/willem/WebIF06/"

</VirtualHost>
```

In this case, the programmer is working on two different versions of the application.

The next step was development of a single CGI procedure to access the web interface, regardless of the web server used. In the current configuration, two web servers were used: one for OSU and one for SWS. These two needed to be joined to form one procedure that was easily adapted to use with WASD (and any other web server). One big advantage with the procedures for OSU and SWS web servers was the significant similarity between them: The code examining and interpreting the URL to set up the environment, to do some consistency and sanity checks, and actual running the report generator were only minor issues. These parts were extracted and placed in separate command files that were then called by the main procedure. For development and testing, a few other command files were created, one of them showing all symbols that are set up by the web server.

Major differences that showed up are based on the difference in servers, and first could be partly solved by determining the existence of the logicals that refered the root directories of the server. That would be fine if either one or the other server was running. However, in this construction, all three servers were active, so using these logicals was not an option. The first possible solution was to define a logical outside the server specifying the server used, but that was abandoned because it was not feasible in this test configuration. In a live system, this solution would require the intervention of a system manager or operator in case the server was switched --- a task that is easily overlooked.

In the end, the solution was to use process information. Since the CGI procedure is run in a subprocess of the server, the name is derived from it. For SWS, its name would contain the string “APACHE”, and for WASD, the name would contain the string “HTTPD”. If neither is true, the server is OSU. That way, some primary setup can be done, as shown in the following example:

```
$       HTTP_SERVER==f$edit(f$GETJPI("","PRCNAM"),"UPCASE")
$       if f$locate("APACHE", HTTP_SERVER) .ne. f$length(HTTP_SERVER)
$       then
$               HTTP_SERVER == "APACHE"
$       else
$               if f$locate("HTTPD", HTTP_SERVER) .ne. f$length(HTTP_SERVER)
$               then
$                       HTTP_SERVER == "WASD"
$               else
$                       if f$locate("HTTP", HTTP_SERVER) .ne. f$legth(HTTP_SERV
$                       then
$                               wrnet "<DNETRECMODE>"  ! Set implied carriage c
$                               mcr www_root:[bin]CGI_SYMBOLS "WWW_" "FORM_"
$                               HTTP_SERVER == "OSU"
$                               URI == P2
$                       else
$                               ! --- foutje-------
$                               err = "Ophalen soort server APACHE/OSU mislukt."
$                               err1 = "Check logical BPS$SERVER_SOFTWARE."
$                               @gate:gate_alert "HTML_TAG"
$                               goto closeMe
$                       endif
$               endif
$       endif
```

The type of web server is also important in the retrieval of server data. For instance, the OSU web server uses a different method than SWS or WASD. For OSU, the URL up to the question mark (“?”) is passed as a second parameter to the CGI procedure, the rest as a symbol. So the full URL needs to be contructed. On the other hand, SWS and WASD get the full URL by symbols set by the server, but the names differ.

For WASD, we found another difference: data embedded in a <FORM> was found to be named differently than the way SWS names the symbols. To be able to retrieve these arguments, an extra translation was found to be necessary, as shown in the following example:

```
$ if HTTP_SERVER .eqs. "WASD"
$ then
$!
$       WWW_ANR                 == "''anr'"
$       if f$type (WWW_FORM_BIJLAGE) .nes. "" then WWW_FLD_BIJLAGE == "''fld_bijlage'"
$       if f$type (WWW_FORM_SOORT) .nes. "" then WWW_FLD_SOORT == "''fld_soort'"
$       if f$type (WWW_FORM_BIJLAGE) .nes. "" then WWW_FLD_KEY == "''fld_key'"
$ endif
```

Otherswise, the program that needs these values is unable to locate them (this image used the same library that is included with WASD, and it worked with both SWS and WASD), as shown in the following example:

```
$ if HTTP_SERVER .eqs. "APACHE"
$ then
$       fullHostName = SERVER_NAME
$       URL_STRING = REQUEST_URI
$       RemoteUser = REMOTE_USER
$!
$ else
$       if HTTP_SERVER .eqs. "OSU"
$       then
$               fullHostName = WWW_SERVER_NAME
$       url_string="''URI'?''WWW_QUERY_STRING'"
$               RemoteUser= WWW_REMOTE_USER
$       else
$               if HTTP_SERVER .eqs. "WASD"
$               then
$                       fullHostName = WWW_SERVER_NAME
$                       URL_STRING = WWW_REQUEST_URI
$                       RemoteUser = WWW_REMOTE_USER
$               else
$                   err = "Unknown HTTP_Server."
$                   err1 = "CHECH SERVER SOFTWARE
$                   @gate:gate_alert
$                   goto exit
$               endif
$       endif
$ endif
```

The same symbol is used to do some preparation of the output.

The preparation for HTML output was found to be different in original procedures. The OSU-based procedure did not contain anything particular for this, but the SWS procedure has the following defined:

```
$ write sys$output f$fao("!/<HTML>")
$!
```

This definition seemed to be a requirement, and removing it caused a “Server Error”, even when coded inside a report. The definition was also found to be a requirement when using the internally present debug mode output. This could again be solved by executing this code only when required, and bypassing it when not applicable.

For XML output, the page needed to be prepared differently for OSU and SWS as well. Both need to have the header defined properly but in a different way, regardless of the server, as shown in the following example:

```
$write sys$output f$fao("Content-Type: text/html")
$ write sys$output f$fao
("!/<!!DOCTYPE html PUBLIC '-//W3C//DTDXHTML 1.0 Strict//EN' 'DTD/xhtml1-strict.dtd' >")
$ write sys$output f$fao
("!/<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='nl' lang='nl'>")
```

The OSU web server requires that the processing is "closed down" before continuing. Again, this is done based on the symbol HTTP_SERVER, as shown in the following example:

```
$ if HTTP_SERVER .eqs. "OSU"
$ then
$    wrnet "content-type: text/html"      ! CGI header
$    wrnet "status: 200 exe ready"
$    wrnet "extra-header: EXEC info, system: ", system, ", prog: ", prog
$    wrnet ""
$ endif
```

With some other server-specific processing, the server-independent CGI procedure was completed and could be used for processing plain HTML, generated reports, execution of executables that produce and handle forms, as well as for displaying any type of data.

However, WASD was found to have one more drawback: If the disk accessed was an ODS-5 disk, it explicitly changed the process parse style to "extended". This caused problems with the report generator, which didn't recognize options passed in lowercase. Investigations revealed that the routine that retrieved the options from the command line used the LIB$GET_FOREIGN library function to retrieve the options and parameters, and checked the occurrence of its parameters – after converting the passed arguments to uppercase. With traditional parsing, LIB$GET_FOREIGN converts the read lowercase data to uppercase, so expected command-line options are indeed found. With an extended parseing style, however, LIB$GET_FOREIGN does not perform this transformation, so the same lowercase option is not recognized. This problem was solved simply by setting the parsing style to "traditional" for all servers.

Once this procedure was running, the translation of special characters in the URL data, as well as the HTML and XML output, needed to be reviewed. This was the only change to be made to the report definitions. Originally, the existence of the logical APACHE$COMMON was sufficient to determine whether translations were required or not, but again, in this configuration it was not applicable. Because the symbol HTTP_SERVER has been set up by the CGI procedure based on the calling server, we can now use that symbol to perform, or bypass, the conversions.

```
$defvar #http_server, symbol
…
$if #http_server = "APACHE" $or #http_server = "WASD"  $then
$translate #url "%20"  "  "
$endif
```

The programmers were now able to change files in their own environments without intefering with other programmers' work, and the redesign of the web interface could be performed separately from other activities.

Once this was set up, work began on the second system. The work was hastened by the breakdown of the one OSU server, which required a new server to be configured. This system was to be used as a demonstration, test, and research system. Only one environment would be active at a time, which made the configuration somewhat less complex.

Again, the web servers are installed off the system disk, in their own directory, as on the development system. However, the root directories are one level deeper, since it should be possible to have multiple versions of a server at hand – though only one version of a particular server can be active at a time, as shown in the following example:

```
APPL_DISK
   WEBSERVER
      OSU
         3.10          ← WEB$ROOT
           OSU structure
      APACHE
          1.3
           SWS structure
          2.1           ← APACHE$ROOT
           SWS structure
          2.1-1
           SWS structure
      HT_ROOT
          9. 1
            WASD Structure
           9.2           ← HT_ROOT:
            WASD Structure
```

In this example, OSU version 3.10, SWS version 2.1 and WASD version 9.2 are active.

In this way, the active version can be easily switched, by stopping the server, redefining the logical referring the root directory, and restarting the server.

Installation off the system disk is required on this system because of its nature: that is, it is destined to be used to test the application on new versions of OpenVMS, as well as to test the web interface under new versions of the web servers. One of the system's disks is kept free for testing alone, so installing a new OpenVMS version would not require a reinstallation of server software, and switching between versions would be as easy as rebooting.

Like the development system, the CGI directory tree is located outside the web server structure, but a link is made in each of them, so here as well, the CGI directory seems to reside directly below the web server. That allows the internal structure of the web interface to be exactly the same as on the development system – no matter from what environment it is copied. (If different environments were required, the construction within each of the web servers would be similar as the configuration on the development system. This however has not been installed at this moment.)

The CGI procedure that was created on the development machine was copied to this machine and tested with all three servers. Changes were applied on the development system and were copied back afterward. Once the procedure was finalized, both the application and the web interface were copied to this machine.

Conclusion

Setting up an environment where multiple developers work concurrently on web applications and are able to debug and test them in a natural way -- using a web server – is not a time-consuming task in itself. Given the abilities of the web servers themselves (virtual hosts on different ports) and of OpenVMS (logicals to define multiple locations, creating aliases for directories), it is fairly simple. Assigning ports and a principal configuration of the default web on each server take the majority of time. Setting up a system with multiple, concurrent web servers is no problem either, as long as each one runs on its own port.

Developing a single, all-purpose CGI procedure for an application that works regardless of the web server used requires more work and requires knowledge of each web server's method of naming symbols, the way it handles logicals and special characters, and, of course, of the web application's requirements. For this study, having two very similar procedures that were already working was a huge advantage; starting from scratch would have caused far more work. Finally, the book by Alan Winston titled *OpenVMS with Apache, OSU and WASD: The Nonstop Webserver,* although an older volume, was still very helpful in our testing scenarios.

Simplification thru Symbols

Author: Mr. Bruce Claremont, Software Migration & OpenVMS Consultant

Overview

DCL procedure development and maintenance can be simplified through judicious use of DCL symbol assignments. This article provides an outline of how to use symbols to improve procedures and the user interface.

Introduction

Many coders develop code with little thought to those that will inherit it or the users that must interface with it. I develop code with two core objectives always in mind:

- Ease of maintenance
- User friendly

Achieving these objectives with DCL is easy with a little advanced preparation. Part of that advanced preparation is using a consistent set of symbols to represent common functions, which is the subject of this article.

Defining Common Attributes

In addition to storing values, DCL symbols provide a nice way to define shortcuts for commonly used commands. Over time I developed a set of global symbols that define commands, values, and VT escape sequences that I commonly employ in DCL procedures. I collected them into a single procedure I named SET_ATTRIBUTES.COM. I invoke SET_ATTRIBUTES at the beginning of each application procedure I write and I use it in most of my home-grown system maintenance utilities.

SET_ATTRIBUTES.COM Procedure

The SET_ATTRIBUTES procedure appears in its entirety in Figure 1. Subsequent sections discuss the procedure components in detail.

Figure 1: SET_ATTRIBUTES.COM Procedure

```
$! SET_ATTRIBUTES.COM
$! Procedure defines standard symbols and escape sequences.
$!
$ IF F$TRNLNM("MSI$ATTRIBUTES") .EQS. ""
$   THEN
$!
$!     Determine processing mode.
$!
$      _BATCH          == 0
$      IF F$MODE() .EQS. "BATCH" THEN _BATCH == 1
$      _INTERACTIVE    == 0
$      IF F$MODE() .EQS. "INTERACTIVE" THEN _INTERACTIVE == 1
$!
$!     Define VT terminal control codes (escape sequences).
$!
$      _BELL[0,8]    == %X7
$      _ESC[0,8]     == %X1B
$!
$      _BLINK        == "''ESC'[5m"
$      _BOLD         == "''ESC'[1m"
$      _CANCEL       == "''ESC'[m"
$      _CEOL         == "''ESC'[K"              !Clear to end-of-line.
$      _CEOS         == "''ESC'[J"              !Clear to end-of-screen.
$      _CLEAR        == "''ESC'[2J''ESC'[1;1f"  !Clear entire screen
$!                                              !and home cursor.
$      _EOS          == "'_ESC'[24;1f"          !Go to end of screen.
$      _REVERSE      == "''ESC'[7m"
```

```
$      _UNDERLINE   == "''ESC'[4m"
$!
$!     Get process information.
$!
$      _PID   == F$GETJPI("","PID")
$      _NODE  == F$GETSYI("NODENAME")
$      _USERNAME == F$EDIT(F$GETJPI("", "USERNAME"), "COLLAPSE")
$!
$!     Define common DCL command symbols.
$!
$      RESET*_TERM :== SET TERMINAL /LINE_EDIT /NUMERIC_KEYPAD
$!
$      IF _INTERACTIVE
$        THEN
$             SAY :== WRITE SYS$COMMAND
$             ASK :== READ SYS$COMMAND /END_OF_FILE=CANCEL_PROCEDURE /PROMPT="
$        ELSE
$             SAY == "!"
$             ASK == "!"
$       ENDIF
$!
$      ASSIGN /NOLOG 1 MSI$ATTRIBUTES
$  ENDIF
$!
$ END_PROCEDURE:
$ EXIT
```

What's With the Underscores?

The underscores you see prefixing many of my symbols are not a necessity. They are a convention I use to avoid conflicts with existing symbols on client systems. For example, a client might be using a symbol called ESC, and I do not want to confuse their ESC symbol with mine, so I prefixed mine with an underscore; i.e., _ESC.

Saving Cycles

The first item in the procedure is a check for the existence of the process logical name MSI$ATTRIBUTES.

```
$ IF F$TRNLNM("MSI$ATTRIBUTES") .EQS. ""
$   THEN
```

The reason for this goes back to the dawn of the computer age when saving processor cycles was important. The symbols created by this procedure generally only need to be defined once per user session. To ensure the symbols get defined, I place a call to this procedure at the beginning of all user procedures. The MSI$ATTRIBUTES check determines if the SET_ATTRIBUTES procedure has already been run. If so, there is no need to run it again. Thus, processor cycles are saved, something which is still beneficial today. You will find the following ASSIGN statement at the end of the procedure that creates the MSI$ATTRIBUTES process logical.

```
$      ASSIGN /NOLOG 1 MSI$ATTRIBUTES
```

Now I can already hear all the VMS cognoscenti groaning that placing the SET_ATTRIBUTES call in SYS$MANAGER:SYLOGIN.COM or the user's LOGIN.COM procedure would eliminate the need to call it from each application procedure and they are right. Placing the procedure call at the beginning of each application procedure was a packaging decision. It allows me to install applications on multiple systems at multiple sites and not have to concern myself with modifying system or user login processes.

Process Mode

```
$!     Determine processing mode.
$!
$      _BATCH          == 0
$      IF F$MODE() .EQS. "BATCH" THEN _BATCH == 1
$      _INTERACTIVE    == 0
$      IF F$MODE() .EQS. "INTERACTIVE" THEN _INTERACTIVE == 1
```

Knowing the process mode is often useful and it is information that only needs to be captured once per session. Knowing whether a process is running interactively or in batch allows intelligence to be built into procedures to determine how they respond to errors and exceptions. For example, a procedure running interactively that encounters an error could check the _INTERACTIVE symbol and upon verifying interactive mode, issue a message to the user terminal and pause for a response. The same procedure running in batch would check the _INTERACTIVE symbol, find it was not running interactively, issue an error message via e-mail, and abort with an error status.

Binary Overlays

```
$      _BELL[0,8]    == %X7
$      _ESC[0,8]     == %X1B
```

Binary overlays provide a handy method to place non-printable characters in a symbol. In my case, two non-printable characters I often use are a hex 7, which sounds the terminal bell, and a hex 1B, or the Escape character, which is used to prefix VT control codes (more on control codes a bit further on). The above example demonstrates how to define the bell and escape characters as DCL symbols, making them easy to access and understand in a DCL procedure.

Terminal Control Codes (Escape Sequences)

When using DCL procedures to interact with users, it is often advantageous to format and highlight text, generating a more attractive, friendly user interface. Doing so makes the user's job easier, a benefit which may have a direct bearing on your continued employment.

VT terminals came equipped with a standard set of control codes that managed data display characteristics. Any VT terminal emulator worth using supports these control codes. The control codes were often called Escape Sequences because they were generally prefixed with the Escape character.

Using the previously defined _ESC symbol, it is possible to create symbols that represent common VT escape sequences. That's what this next section of DCL code accomplishes.

```
$      _BLINK        == "''ESC'[5m"
$      _BOLD         == "''ESC'[1m"
$      _CANCEL       == "''ESC'[m"
$      _CEOL         == "''ESC'[K"              !Clear to end-of-line.
$      _CEOS         == "''ESC'[J"              !Clear to end-of-screen.
$      _CLEAR        == "''ESC'[2J''ESC'[1;1f"  !Clear entire screen
$!                                              !and home cursor.
$      _EOS          == "''_ESC'[24;1f"         !Go to end of screen.
$      _REVERSE      == "''ESC'[7m"
$      _UNDERLINE    == "''ESC'[4m"
```

Process Information

I like having static process information on hand. I often incorporate it into things like temporary file names and informational messages. By creating symbols containing the data in SET_ATTRIBUTES, the lexical functions need only be run once, saving more of those precious processor cycles. The _PID, _NODE, and _USERNAME symbol definitions provide examples of this type of information capture.

```
$      _PID   == F$GETJPI("","PID")
$      _NODE  == F$GETSYI("NODENAME")
$      _USERNAME == F$EDIT(F$GETJPI("", "USERNAME"), "COLLAPSE")
```

DCL Command Symbols

A favorite use for DCL symbols is to create short cuts for long DCL commands. Take a look at the following examples:

```
$!     Define common DCL command symbols.
$!
$      RESET :== SET TERMINAL /LINE_EDIT /NUMERIC_KEYPAD
$!
$      IF _INTERACTIVE
$        THEN
$          ASK :== READ SYS$COMMAND /END_OF_FILE=CANCEL_PROCEDURE /PROMPT="
$          SAY :== WRITE SYS$COMMAND
$        ELSE
$             ASK == "!"
$             SAY == "!"
$       ENDIF
```

The RESET symbol provides a quick way to issue a long SET TERMINAL command. The ASK and SAY symbols provide short, descriptive commands that make both writing and reading DCL procedures easier.

Note the use of the _INTERACTIVE symbol. Whether the process is interactive determines how the ASK and SAY symbols are defined. The commands represented by ASK and SAY are useful in an interactive procedure, but not in a batch procedure. Hence, in a batch procedure the symbols serve to define the lines they reside in as comments.

Putting it Together

Everything discussed thus far is basic DCL. Put it together and you have a mechanism to simplify DCL code development and maintenance. The following procedure provides an example of how the symbols defined in SET_ATTRIBUTES can be used. The procedure checks for the existence of a user-supplied file, prompting for a file name if one is not provided. The symbols created in SET_ATTRIBUTES help simplify and clarify the DCL code in this procedure.

```
$! CHECK_FILE.COM
$!     Check file specified in P1 to ensure that it is valid.
$!     If P1 is null, prompt user for a file name.
$!
$!     On Entry:
$!            P1 - File name             (optional)
$!            P2 - Prompt string         (optional)
$!            P3 - Default extension     (optional)
$!     On Return:
$!            FILE$ - Validated file designation.
$!
$      ON ERROR THEN GOTO ERROR_TRAP     !Error trap.
$      ON CONTROL_Y THEN GOTO ERROR_TRAP !User abort trap.
$      STATUS = 1                        !Default exit status value.
$      @SYS$UTILITIES:SET_ATTRIBUTES.COM !Set terminal & process attributes.
$!
$      FILE$ == ""
$      PROMPT = "File> "
$!
```

```
$ START:
$      IF P1 .EQS. ""
$        THEN
$             IF .NOT. _BATCH
$               THEN
$                    SAY ""
$                    IF P2 .NES. "" THEN PROMPT = P2 + "> "
$                    ASK "''PROMPT'" P1
$               ENDIF
$             GOTO START   !Check file name.
$       ENDIF
$!
$!     Apply default extension if specified and the specified file name
$!     doesn't already have one.  If a file designation ends in a period,
$!     treat that as a valid extension and do not apply the default extension.
$!
$      IF P3 .NES. ""
$        THEN
$             IF F$EXTRACT(F$LENGTH(P1) - 1, 1, P1) .NES. "."
$               THEN
$                    IF F$LOCATE(".", P3) .EQ. F$LENGTH(P3) THEN -
                       P3 = "." + P3
$                    EXT = F$PARSE(P1,,, "TYPE")
$                    IF EXT .EQS. "." THEN P1 = P1 + P3
$               ENDIF
$       ENDIF
$!
$!     Ensure file exists.
$!
$      IF F$SEARCH(P1) .EQS. ""
$        THEN
$             SAY _BELL, "ERROR - File ", _BOLD, P1, _CANCEL, " not found."
$             P1 = ""
$             GOTO START
$       ENDIF
$      FILE$ == F$PARSE(P1,,, "DEVICE") + F$PARSE(P1,,, "DIRECTORY") -
              + F$PARSE(P1,,, "NAME") + F$PARSE(P1,,, "TYPE")
$!
$ END_PROCEDURE:
$      EXIT 'STATUS' + 0*F$VERIFY(VERIFY)
$!
$ ERROR_TRAP:
$      SAY _BELL
$      SAY "An error or <Ctrl^Y> has aborted this procedure."
$ CHECK_POINT:
$      IF F$MODE().NES."BATCH"
$        THEN
$             ASK "'_BELL'Enter 0 to exit the procedure: " CHK
$             IF CHK .NES. "0" THEN GOTO CHECK_POINT
$       ENDIF
$ CANCEL_PROCEDURE:
$      STATUS = 3
$      SET TERMINAL /LINE_EDITING
$      GOTO END_PROCEDURE
```

Conclusion

Using DCL symbols wisely provides the means to simplify your DCL procedures while enhancing their functionality. A procedure like SET_ATTRIBUTES also provides a single point of reference to make global changes to commonly used symbols, which is yet another maintenance benefit. As usual, a little forethought can go a long way towards improving the end product.

Thanks for sticking it out to the end and reading the entire article. Your comments and feedback are welcome. You will find contact information at the *Contact Us* link on the MigrationSpecialties.com web site.

For more information

More on DCL development:

- *Simplifying Maintenance with DCL*,
  http://h71000.www7.hp.com/openvms/journal/v9/simplifying_maintenance_with_dcl.html

Real world DCL examples:

- *ODS-2/ISO-9660 CD Creation*
  http://www.migrationspecialties.com/pdf/ODS-ISO.pdf
- *Using OpenVMS to Meet a Sarbanes-Oxley Mandate*
  http://www.migrationspecialties.com/pdf/Using%20OpenVMS%20to%20Meet%20a%20Sarbanes-Oxley%20Mandate2.pdf

VT Terminal Information (with thanks to Ian Miller for the links)

- http://www.vt100.net/
- http://www.cs.utk.edu/~shuford/terminal/dec.html

Taking T4 to the Next Level

Steve Lieman, Performance Architect, TrendsThatMatter

BIO

Steve has made hands-on practical use of T4 from its earliest beginnings. He actively employs T4, TLViz & CSVPNG as essential ingredients of his performance work for TrendsThatMatter. Steve's audio segments on T4 technical tips are a regular feature of the VMS Podcast Network (VPN) program and Steve wrote the first in-depth article on T4 that appeared in Volume 3 of the OpenVMS Technical Journal. The easy reader version of the article (just created recently) is a great way to begin learning about T4 and to get a sense for the visual nature and the power of this approach.

Steve was a member of the OpenVMS Engineering Performance Group for 11 years and he has more than 25 years of enterprise system performance management experience. Steve has been a regular speaker about T4 at every OpenVMS Boot Camp and at the popular OpenVMS Technical Updates in Europe and in the United States. He has delivered seminars and hands-on workshops on practical performance management at more than two dozen locations in the U.S, Canada, Europe, and the Pacific Rim. Steve also participated on the Transaction Processing Council in its formative years while it was developing and defining the TPC-C benchmark. You can learn more about Steve at TrendsThatMatter.com.

Overview

It is hard to believe that the T4 approach has now been around for almost seven years, and that year by year it has achieved wider acceptance while incrementally improving its underlying capabilities. This is easily seen in the long list of enhancements added to the T4 collector, the large number of productivity changes for TLViz and CSVPNG, and the step-by-step increase in the number of T4 compatible collectors, now totaling more than 20. We have seen time and again that the T4 approach encourages collaboration while doing performance work.

This article briefly looks back at T4's origins and where it has been, and then projects the likely trajectory it will take in the years to come as it builds even further improvements onto an already solid foundation. This article also provides you with references to all the latest T4-related resources and to where you can turn for help and advice so that your OpenVMS performance work can achieve the maximum benefit from this approach.

T4 origins – our time is our most precious resource

T4 began its life seven years ago during a large-scale benchmark just before the release of the AlphaServer GS160 system. Tom Cafarella (OpenVMS Engineering) and I were assisting an important customer compare the GS160 performance to that of the largest AS8400 system available.

The benchmark was especially complex because of its use of captured log data from a live production environment to drive the load. Each test showed highly variable behavior during its brief, 30-minute run – a far cry from steady-state industry-standard benchmarks, which prove much easier to analyze.

We were using the OpenVMS Monitor utility (MONITOR) to watch OpenVMS performance while the customer was using their own software logging program to measure and report on application response time and throughput.

The customer wanted to test many different variations of CPU count, memory size, disk configuration, tuning parameters, and so on. However, the results we were getting were hard to understand and even harder to explain. We were finding that it took 1 to 2 days to investigate and analyze even a single head-to-head comparison between the GS160 and the AS8400 systems. The customer hoped to test at least a dozen different options per day in order to complete the work in the allotted time. Needless to say, Tom and I were under a lot of pressure.

Necessity is the mother of invention

The T4 approach was born into this cauldron of activity. Tom Cafarella wrote a DCL script that started with a `Monitor.DAT` file and ended up producing a comma-separated value (CSV) text file. The text file contained what Tom and I considered the 30 most important OpenVMS performance factors. These included CPU idle, kernel mode time, interrupt mode time, mpsynch time, buffered I/O, paging I/O, and direct I/O. Tom's code replaced a manual process we had been using that took us hours or even days to finish and that was not nearly as complete. We could now get a timeline view of how all 30 of these important factors varied during the complex course of the benchmark. And we could now complete this first level analysis within a few minutes after the end of the test. This development changed everything.

The CSV file that we created later became known as T4-style format. The file contained a table that had a few header rows, including one where each performance factor was listed by name, followed by a set of rows, with exactly one row for each sample time period. Each column of the table represented a single performance factor. Because it was in CSV format, and because all the data for a single factor was contained in a single column, we could open this file with Microsoft® Excel software and then, with about six mouse clicks, plot the trend lines for that factor.

Time is of the essence

This time savings achieved with Tom's DCL script made a huge difference in our ability to analyze these complex benchmark test runs in a reasonable amount of time. Whereas it had been taking days per run, we were now able to discover important lessons about a run within an hour or two of the end of the test. We could then use that information for feedback into the ongoing process, thereby enabling more focused selection for the next round of benchmark tests.

The T4 name

In honor of Tom's yeoman and timely efforts to create the DCL conversion script, the original name "T4" was meant to represent Tom's Terrific Timeline Tool. Since then, as T4 evolved and a new compiled version of the extractor was constructed, the T4 name came to mean Total Timeline Tracking Tool. Given how much time this approach has saved us over the years, we might think of renaming it yet again to something like Time-saving Trend Tracking Tool. Whatever name we give it, because of its significant ability to save our time, the T4 approach has completely transformed the way we carry out performance work on OpenVMS systems.

T4 - the early years

Within a very short time after its first use, the OpenVMS Engineering organization began to apply T4 to real-life production performance problems. Just like the original benchmark, T4 was perfect in these situations because production systems rarely show steady-state behavior. To deal with the ongoing variability of each important factor, it is essential to first grab a time-series view of how that factor changes over time, and then to be able to examine it visually. The T4 approach evolved rapidly in response to a spike in performance issues related to the introduction of the GS160 system.

The extraction of MONITOR data was rewritten by Ian Megarity (OpenVMS Engineering) as compiled code, and many new performance factors were included, such as detailed data about each CPU and about each disk.

New collectors that generated output in the T4-style CSV format were then added that captured important performance factors not available with MONITOR. These included a TCP/IP collector called TPCMON (developed by Matt Muggeridge) and a dedicated lock manager collector (developed by Ian Megarity). The collectors were bundled into a new DCL script (that we called T4)

so that MONITOR and all the associated collectors started and stopped at the same time, and all used the same sampling rate.

Another new tool called Append Record (APRC) was created by Ian Megarity to consolidate the CSV files generated by the different collectors into a single, larger CSV file (known as the *COMP.CSV file) containing all the factors from different collectors.

Saving more time with TLViz. Another magic piece of puzzle was completed when Ian Megarity created a new timeline visualization tool called TLViz (TimeLine VIsualiZer). TLViz made it incredibly easy to visualize the trend data of each individual factor. And TLViz made it just as easy to view any interesting combination of factors. Our experience with this tool proved that we could analyze data at least 5 times, and often 10 times, faster than when we read the same CSV file into Microsoft Excel. Kevin Jenkins (from OpenVMS Engineering) suggested the idea of opening *before and after* CSV files and having TLViz overlay the data. Ian Megarity implemented version 1 of that idea literally overnight, changing forever and for better the way OpenVMS performance engineers carried out their analysis and comparison of complex production systems.

Because of its visual nature, TLViz also proved to be amazingly good for doing collaborative work with a group of analysts, as well as being a most effective tool for demonstrating the results of analysis to both technical and non-technical audiences. If you care about performance but you have not yet tried TLViz, I guarantee that you are in for a treat when you first experience its features and its built-in time-saving capability.

To give you a flavor of TLViz and its powerful Before and After feature, Figure 1 shows a visual example from the more than two dozen charts contained in the simplified version of the original T4 article in the OpenVMS Technical Journal. While these kinds of graphs in a static format are useful, we have found that direct use of TLViz can be even more powerful. We have also discovered that interactive use of TLViz to present results turns out to be much more effective than creating static reports.

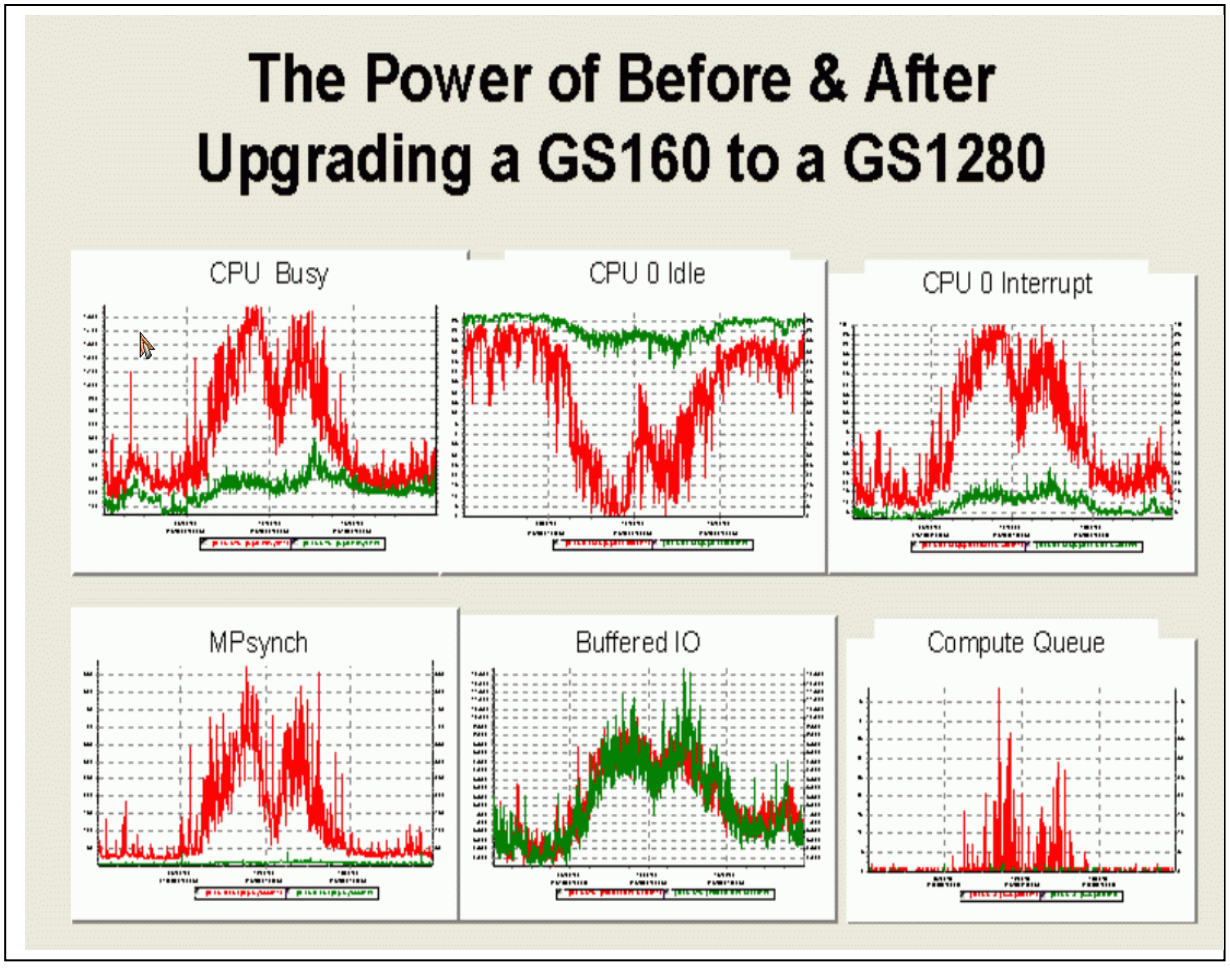


Figure 1 – Type and extent of changes as reported by TLViz

Automating repetitive work with CSVPNG. Building on the T4 and TLViz foundation, Pat Moran (HP Support, Ireland) invented yet another new tool, called CSVPNG. Among other things, this tool can automatically convert T4-style CSV files to PNG graphics format. This has proved to be the perfect complement to TLViz in that it allows the analyst or system manager to program repetitive performance-analysis steps using a DCL script that includes one or more CSVPNG command-line commands. Because CSVPNG runs on OpenVMS Alpha and HP Integrity servers, you can add your customized CSVPNG string to the tail end of each T4 collection session. This is exactly the kind of job that fits well with the skill set of trained OpenVMS system managers.

T4 successes and improvements

As the years rolled by, the T4 approach continued to evolve, often at a rapid pace. More and more OpenVMS customers pressed it into service on their most important and largest-scale systems. It proved helpful right out of the box and even more so when small custom enhancements were added. The following sections briefly highlight some of the successes and improvements that have been added.

T4 collector improvements. The first T4 involved manually running MONITOR and then manually running the T4 DCL extractor code to produce a CSV file that could then be read into Microsoft Excel. Very early on, Ian Megarity combined these two steps and added an automatic way to run the T4 collector once and instruct it to turn on what might be called History Mode. In this mode, the collector continued measuring and creating T4-style CSV files, once per day, 365 days a year. With History Mode turned on, we were assured that we would always have the baseline data we needed for comparison whenever anything changed or whenever a new problem arose.

The T4 DCL script continued to add new collectors and to integrate them as part of a standard T4 collection. These now include a network adapter collector for each network adapter, a basic XFC collector, and a Fibre Channel Monitor collector named FCMON. FCMON provides detailed response time and kilobytes per second KB/s rates for each disk from the OpenVMS server point of view.

Optional collectors for spinlock data and for TDC data are also now available with the prerelease version of T4 version 4.2.

TLViz improvements. Thanks to the continued efforts of Ian Megarity, TLViz experienced a steady progress of new features that made it easier to use and that allowed the analyst to dig deeper and deeper in reasonably short periods of time. Some of the most useful features added include moving averages, calculation of the ratio of two factors, scatter plots and correlations between a pair of factors, area charts and 100% area charts, and bar charts for cases with relatively few data points.

TLViz added some very powerful capabilities for quickly navigating CSV data sets that contain a large number of different performance factors. This meant we were able to apply its time-saving visualization to more and more complex situations, including cases where there were literally thousands of separate factors to consider.

Recently, Pat McConnell (OpenVMS Engineering) has added some terrific new features in version 2.0-1. There are now more "radio" buttons on the bottom left of the screen that simplify and speed up some common and important operations that arise frequently during analysis situations.

CSVPNG improvements. In the capable hands of Pat Moran, CSVPNG has continued to evolve, improve, and add new functional capability. CSVPNG now includes more than 30 separate command-line directives that work individually or together to manipulate, automate, visualize, and report on T4-style CSV files. CSVPNG is particularly powerful now for trimming down a huge data file by selecting exactly the performance factors you are most concerned with and further selecting the precise time periods that interest you. CSVPNG contains multiple options for combining T4-style CSV files in different and useful ways. These include assembling a week or month of data into a single file and combining data from multiple nodes in a cluster.

CSVPNG can also input and apply its magic to an increasing number of files of slightly different formats, including tab-separated files from the Linux COLLECT-L utility, and outputs from the Microsoft Windows® PerfMon utility.

T4-compatible collectors and extractors

A growing number of T4-compatible collectors and extractors continue to appear. Recently, we have seen the introduction of the VEVAMON collector that runs on OpenVMS and collects detailed storage performance factors from EVA5000 and EVA8000 systems. We also have heard that a new approach for extracting HP's XP storage data and transforming it into T4-style format is also in the works.

Other T4-style collectors have been built for Rdb, Oracle®, additional OpenVMS memory factors, the Collect utility for Tru64 UNIX®, and NFS.

With its latest releases for the EVA, HP Storage now includes instructions for how to convert EvaPerf data into T4-style, TLViz-compatible format.

These developments are part of a wave of growing appreciation for the idea that the T4 approach is not just for OpenVMS. The conclusion: When data is organized in T4-style files, time-saving, enhanced productivity, improved depth-of-analysis benefits, strengthened collaboration, and greater ease of reporting are immediately available.

The latest T4 news

New versions of TLViz and CSVPNG are now available. For more information, see the links at the end of this article.

A number of developments are underway to build additional analysis tools that will accept T4-style data as input and provide additional analysis benefits not currently available. We will be reporting about these on the TrendsThatMatter blog as soon as they become available.

T4 future trajectory – building on the foundation

The T4 approach has proved that it works and that it saves huge amounts of time for the analyst. It has proved that it benefits OpenVMS performance work. It has also proved that any trend data can be assembled into a T4-style format and thereby tap into T4's time-saving benefits.

A solid foundation is now in place. A substantial percentage of mission-critical OpenVMS systems now in production operation have been collecting T4 data and building performance histories that go back one or more years.

New collectors, when they became available, have proved to be easily integrated into the picture because the basic T4 approach is inherently extendable. This is a huge area of opportunity because we have found that while "out of the box" benefits are good, the benefits of local customization are even better.

Even after 7 years of T4 development and the existence of more than 20 collectors, some vital metrics still are not regularly collected in a form that allows them to be viewed easily as trend data or integrated with existing performance data. For example, there is no easy way today to observe CPU chip-level behavior such as Data Cache Miss Rates as they change over time. We are certain that once such a chip-level trend data collector becomes available, it will open the door to identifying opportunities for radical performance improvements. For more information about this topic, see Extending the T4 Approach to New Territory.

In addition, the users of TLViz and CSVPNG continue to come up with good ideas for how to make these two important tools even more useful and powerful and even better at saving our precious time.

For those who want to extend the life of their current AlphaServer systems, the T4 approach can provide you with the data and analysis tools that help you push your systems as far as they are capable of going.

For those moving toward using the new powerful, Dual-Core Intel® Itanium® 2-based Integrity server systems, the T4 tool set can help you get ready for that transition, make sure that you know how much spare capacity you have and how fast you are using it, assist you in sizing your new system, and validate that the new system delivers the performance level you expected.

For those who are thinking of consolidating many current OpenVMS systems into a new, larger system and of taking advantage of the growing virtualization capabilities, your T4-based performance history on your existing systems will prove to be essential input data. When dealing with consolidation, CSVPNG's ability to combine data from separate CSV files will surely prove to be an invaluable time saver that can help you achieve the best possible results.

T4 resources

The following links show you where to find the most important T4-related information and where you can turn for help and advice along the way. If you are interested in The T4 Universe, these resources will help you both get started and keep up with the latest developments.

If you have T4 issues or questions: please send email to and to T4@TrendsThatMatter.com.

Want an advance copy of T4 V4.2 with the option for Spinlock Tracing and TDC collection? Send your request to

Struggling with a particularly complex performance situation where the T4 data just doesn't add up and you are looking for a quick second opinion? You can send email to T4 Mini-Evaluation.

Would you like to know more about the TLViz Formatter that you can use with EVAPerf data? You can find details in the HP StorageWorks Command View EVA user guide.

Looking for the current version of the T4 collector for OpenVMS and other T4 approach tools? You can download these from the HP T4 Home Page. Also see the HP T4 FAQ.

The latest T4 & Friends news can be found by checking on the TrendsThatMatter blog.

You will find some useful downloads of new versions of T4-style tools at the TrendsThatMatter download page.

New to T4? The TimeLine Collaboration article from the OpenVMS Technical Journal, Volume 3 is a good source of detailed information about the T4 approach to performance and about all the many people who have played a role in its development.

A streamlined version of the TimeLine Collaboration article (Quick Intro to T4) is a great place to start learning about T4. This work is inherently visual and is most understandable when you look at the actual performance trend data with tools like TLViz and CSVPNG. This quick introductory version includes many easy-to-understand examples.

Interested in using TLViz or CSVPNG to examine the rich range of T4-style data available from the many different collectors? Please send mail to: T4-style data samples.

Rather learn more about T4 by audio? You can check out the full set of T4 segments from the VMS Podcast Network (VPN) at: T4 Audio Segments.