Internet Technologies for OpenVMS

Ken Moreau
Solutions Architect, OpenVMS Ambassador, MCSE

Overview

This paper discusses Internet technologies and web services available on platforms other than OpenVMS, and shows how these technologies and services work in an OpenVMS environment. The topics discussed in this paper include Java, XML, Enterprise Application Integration (EAI), database connectivity, adding graphical front ends to existing terminal-based applications, and web browsers and servers.

Introduction

You have a history with OpenVMS. You have years and possibly decades of experience and knowledge embedded in the data, applications, business logic, and processes that are currently satisfying your business needs better than any other operating environment. You want to protect not only this business logic, but also the skills you have developed that built this business logic.

But there is a lot of pressure to move into the "brave new world" of the Internet. The question becomes, how do we most effectively use the best features of each world?

We do it by taking Internet technologies available on other platforms and making them available on OpenVMS. We do it by simply taking the data, applications, business logic, and processes that are currently running very well on OpenVMS, and exposing them to the larger Internet world, so that, for example, the UNIX client running the browser doesn't even notice it is using OpenVMS. So that the Excel spreadsheet on Windows, or the program running on Linux, doesn't even notice that the data they are fetching is coming from OpenVMS. So that the business-to-business (B2B) application that your business partners demand that you use simply works, and they don't even notice that they are running programs on OpenVMS.

And we do it by having your OpenVMS applications transparently connect to the rich variety of information available on the Internet, so that you can fetch data from another system, and you don't even notice that it is not running OpenVMS. This gives you transparent and seamless integration, while keeping all of the advantages of your OpenVMS systems: security (OpenVMS is absolutely immune to virtually all of the viruses we have seen), reliability, disaster tolerance, scalability, and so on.

Personally, this integration means you can continue to use the skills you already have and add some new ones to help you keep up with the latest and greatest technologies.

Java

Java was invented by Bill Joy and his team at Sun Microsystems, and represents one of the most exciting technologies available today. One of the reasons that Admiral Grace Hopper helped to specify COBOL back in 1959 was so that programs could run on every computer system available at the time. COBOL didn't quite fulfill that promise, and other languages like FORTRAN, C, C++, and BASIC still require work in order to compile and run on every operating system with every compiler from every vendor. But Java goes further than any other language in delivering on that promise. You truly can write Java programs once and run them everywhere. And that includes running them on OpenVMS.

Java is a programming language, with source code that is fairly easily readable. But Java is also a runtime environment with a lot of supporting infrastructure. The infrastructure is the key to Java's portability, because each operating system implements the infrastructure with exactly the same interfaces. Java programs call those interfaces and simply don't care what the underlying operating system is doing.

The two components of the Java environment are the Java Development Kit (JDK) and the Java Run-time Environment (JRE). The JDK contains all of the things that you would expect to be able to develop, debug, and deploy applications. The JRE contains only those components needed to deploy applications.

The combination of the two components comes in the three flavors: the Java 2 Platform Standard Edition (J2SE), which usually runs in client machines such as workstations or PCs, the Java 2 Platform Enterprise Edition (J2EE), which runs on the servers the client machines connect to, and the Java 2 Platform Micro Edition (J2ME), which runs on handheld devices that don't have a hard disk or a keyboard, such as an iPaq running PocketPC.

The J2SE, J2EE, and J2ME all contain a Java Virtual Machine (JVM), which all Java programs run within. Different operating systems implement the exact same virtual machine, so the fact that there are different operating systems underneath the virtual machine is hidden. OpenVMS can implement Java compilers and the JRE with a JVM, and all of the Java programs in the world just work.

A virtual machine is good, but you know how programmers are: they want to extend the base functionality. The Java specification includes the ability to add new functions to the JRE by creating "beans," which are simply run-time modules that the Java programs can then call, just as if they were specified in the original J2xE. Java programs simply state during the installation that they require this or that set of Java beans, enclosed in this or that Enterprise Java Bean (EJB) library. Anyone who wants to can write a Java bean, and can then supply it to the world, just by publishing it and getting other developers interested in using it.

Java programs also want to access data, and there are many database formats to choose from. In the same way that the Open Data Base Connectivity (ODBC) interface allows programs in any language to access databases of different formats, the Java Data Base Connectivity (JDBC) APIs are a set of Java beans that allow Java programs to access databases of different formats. Each database vendor, and a few other companies, have supplied beans that let Java programs talk to their specific databases, so all Java programs can access those databases without having to worry about the details of the database formats.

There are many APIs that have been added to the base specification. To understand the APIs, the first thing you need to know is that the developers are acronym-happy.

Here is a (very) partial list of the Java APIs:

- J2EE – Java 2 Platform Enterprise Edition. The base set of APIs and run-time utilities for every Java program running on a server.

- JAAS – Java Authentication and Authorization Service. If you want to authenticate your users, similar to logging in to OpenVMS and checking your password against the username and password in the SYSUAF.DAT file, you use JAAS.

- JAR – Java Archive, similar to UNIX 'tar' (tape archive) format.

- JCA – Java Connector Architecture, which defines the APIs to allow programs running on the application servers to connect to the traditional, legacy production systems running the core of the business, such as Enterprise Resource Planning (ERP) systems like SAP, or even some of the OpenVMS systems that you develop. The point here is to connect to information that is not in relational databases, because JDBC takes care of that problem. We will see more about this later.

  Developers couldn't let a perfectly good acronym go to waste by only using it for one thing, so JCA also stands for the Java Communications API, which lets Java communicate with voice-mail, FAX, and smart card devices. When you see this acronym, you will just have to figure out for yourself which one they mean.

- JDBC – Java Data Base Connectivity. Similar to ODBC, but callable by Java programs to access data from a variety of databases and systems.

- JDK – Java Development Kit. Development is done with the JDK. Sun invented Java and still sets the specifications for it, and everyone, including HP, licenses the tools from Sun. Sun wants to encourage the use of Java, so the license terms are reasonable.

- JMS – Java Message Service. If one Java program wants to send a message to another Java program, it uses JMS. This shows the power of Java to hide the underlying operating system. On UNIX we might implement this with pipes. On OpenVMS we might implement it with mailboxes. If we are communicating between systems we might use TCP/IP sockets. JMS works either synchronously or asynchronously, using either point-to-point connections or publish/subscribe methods. The Java program doesn't care, it simply uses the JMS interface and it just works.

- JNDI – Java Naming Directory Interface. This allows you to talk to Common Object Request Broker Architecture (CORBA), Lightweight Directory Access Protocol (LDAP), X.500 or Microsoft Active Directory directories, and register your own names and look up other names in your enterprise name directory.

- JRE – Java 2 Platform Run-time Environment. This contains the virtual machine and the set of APIs common to all Java implementations.

- JSP – Java Server Page. Sun observed Microsoft's Active Server Pages, so Java has Java Server Pages. Java Server Pages technology allows web developers and designers to rapidly develop and easily maintain information-rich, dynamic web pages that leverage existing business systems. JSP enables rapid development of web-based applications that are platform independent, and separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

- JTA – Java Transaction Architecture. Computer systems are important, and they are very reliable: OpenVMS is extremely reliable. But sometimes they do fail (OpenVMS less so than any other platform), and frequently transactions are distributed among several different systems running different operating systems. People use an external transaction monitor like Tuxedo to watch over the state of transactions inside the entire environment. Java communicates with these and monitors the entire distributed transaction with the JTA.

- JPDA – Java Process Debug Architecture. How many people here write perfect code the first time? For the rest of us, Java specifies a system-independent way to debug Java applications, using the JPDA. It is not a debugger itself, but provides the tools to build debuggers and to run full debugging consoles on remote systems, across a network.

- JRMI – Java Remote Method Invocation, JRMI allows you to call programs and invoke methods outside of the system you are running.

The intent of all of these Java APIs, and the dozens more that I didn't list, is just like the JRE or the JDBC specification: to have a single way of doing something across all operating systems. You call the specific function using the right interface, and your programs that were developed on, for example, a UNIX flavor or Linux, simply run on OpenVMS.

There are many more Java APIs. For more details, enter "Java" in your favorite search engine, or go to http://www.java.sun.com/products. This is the definitive website for all Java work, and is the high level page that will lead you to more information about all of these technologies and a lot more.

How does all of this fit together? There are two major pieces: the client side and the server side, with the server side broken down into several different components.

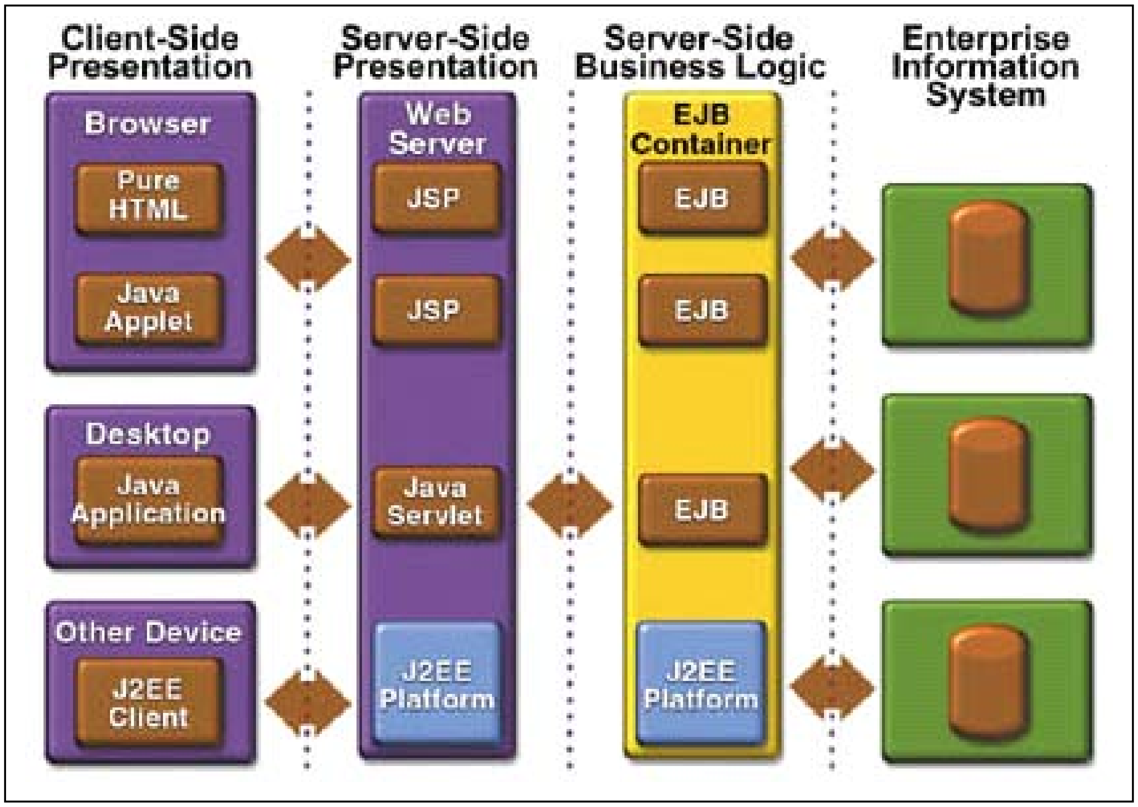
The client side includes the things that we are most familiar with, primarily browsers. The applications can be written in pure HTML or in Java. Some of these are not full applications like we understand them; they are smaller and less rich in functionality. As such, we call them applets instead of applications. You can also write sophisticated applications, because Java is a full 3rd generation language (3GL). You can even use devices other than PCs, such as iPAQs, web-enabled phones, and so on, using the J2ME client.

The server is divided into the presentation layer and the business logic layer. If you have ever used DECforms or FMS to develop applications, you understand the distinction between presentation and business logic.

The presentation layer is written in Java and runs inside a web server, which is the JRE and includes the JVM referenced earlier. Just like before, we can have simple pages written as Java Server Pages (JSP) and simple applications written on the server, called servlets. All of this is written in Java, and depends on the J2xE server environment.

The business logic can be written in any language. (We will talk later about how to integrate your existing programs into this layer, but for the moment we will talk about business logic written in Java.) These programs are created as enterprise Java beans, and are just like the enterprise Java beans that come as part of many packages. Whether you write the bean or someone else does, it is all simply part of the environment, and run in the same environment.

On the back end is the data itself, which is accessed by the JDBC interfaces discussed earlier.

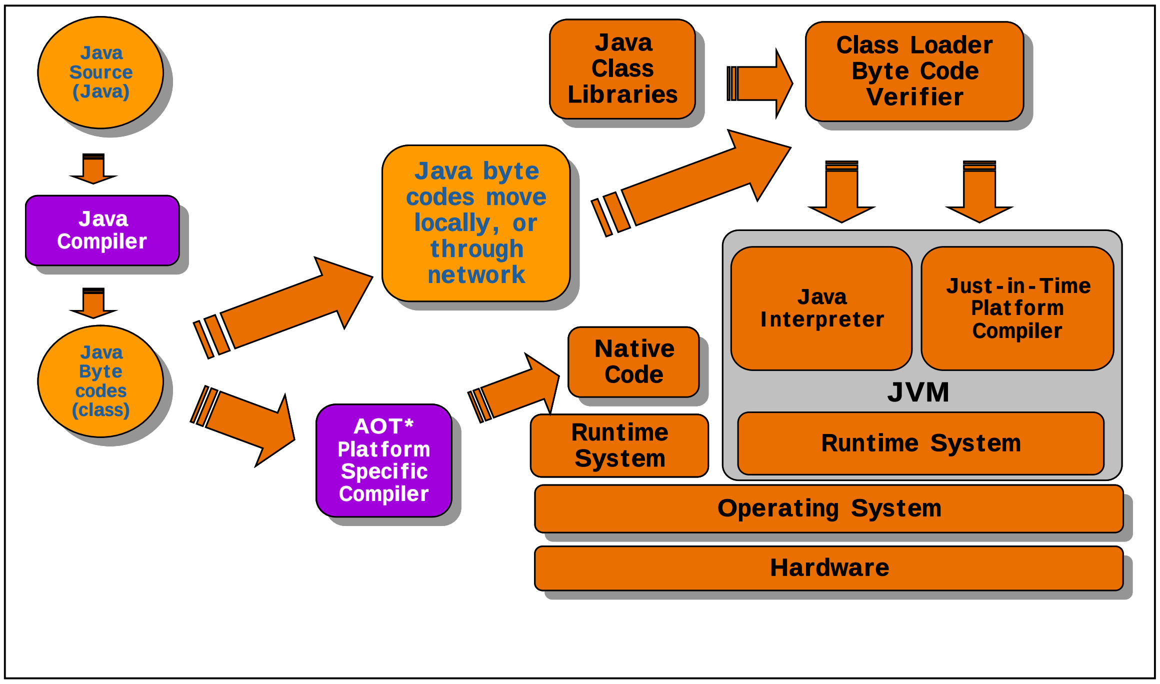


How do you create these beans? In much the same way you do for other languages, but there are a few differences.

Start with source code and run it through the compiler, the same as in any language like FORTRAN, COBOL, or C/C++. The result of this isn't object code, but a system-independent byte stream, which you then send to the system that will run the Java program. This program can be an applet, a servlet, or a full application.

Once the byte stream is on the target system, it is combined with the class libraries and fed to the JVM. You can run the byte code directly through the interpreter, or execute the Just-In-Time (JIT) compiler that is specific to the operating system and hardware. This compiler takes the byte code and turns it into a real native application for that specific O/S and hardware combination.

The interpreter is slower but more flexible, while the JIT compiler makes the application run faster but takes longer to compile, so there is a higher startup cost. One way around this is to use the Ahead-of-Time (AOT) compiler, which pays this cost at compile time. Think of this as a standard compiler and linker, which compiles Java instead of C/C++ or FORTRAN.



Some people want more than a compiler to build applications: they want an Integrated Development Environment (IDE). The IDE for Java is called NetBeans. This was developed by a group of students who needed a Java IDE, and is now distributed by Sun.

Third parties can create their own branded distributions of the NetBeans IDE including additional modules they've written, or can simply make those modules available either commercially or for free to plug into existing distributions of NetBeans. Sun One Studio, OptimalJ and ObjectAssembler are three examples of extensions to the base NetBeans IDE.

There is an "update center" in the IDE that allows modules to be downloaded and installed into a running copy of the IDE. http://netbeans.org hosts an update center server for non-commercial modules. Sun operates an e-commerce enabled update center that allows third parties to sell and distribute commercial modules online.

The NetBeans Update Center has an OpenVMS section for our plug-ins, which are also available from the OpenVMS web site. The C/C++ compiler support is tailored to the OpenVMS C/C++ compilers and includes source colorization, code formatting, source code compilation (but not linking), handling of messages that come from the compiler to update the source window, and provides DCL command procedure execution either in the same window or a separate terminal window.

IBM developed their own IDE framework called Eclipse (aimed at Sun). In 2002 they made it open source, and specifically didn't invite any NetBeans people to the party. Eclipse uses the Standard Widget Toolkit (SWT) written in native code for each platform. IBM does not port this to OpenVMS, so IBM's Eclipse will not be available on OpenVMS. Don't confuse this with HP's Eclipse, which is the Java back-end that HP wrote.

What does all of this mean for OpenVMS?

Everything we talked about is available on OpenVMS: Java, the development environment, J2EE, NetBeans, JDBC -- it's all on OpenVMS. Every time Sun releases a new version, or someone releases a new NetBean or EJB library, we release them on OpenVMS. And I say "we" inclusively, because a lot of this software comes from places outside of HP. We depend on our software partners like BEA, Attunity, and Apache. They are releasing their code on OpenVMS at the same time that they are releasing it on every other platform.

The bottom line is: Java is the same on OpenVMS as it is on every other operating system. Anything you can do in Java on HP-UX, Windows, Linux, AIX, or even Solaris, you can do on OpenVMS.

There are some differences between Java on UNIX and OpenVMS -- some of which you would expect, and some of which are surprising:

- Most of the development work for Java and the beans is done on UNIX systems. This means that the developers frequently require the use of shell scripts (what we call command procedures) to carry out simple tasks. Because they are on UNIX and/or Windows, these are not written in DCL. When porting a Java program to OpenVMS, you need to convert the UNIX shell scripts to DCL.

- For classic OpenVMS people, UNIX has some bizarre requirements for file names: upper and lower case, strange characters, really long names, lots of dots, and so on. This means that those Java applications that use file names that aren't supported under ODS-2 must be installed on an ODS-5 disk. This is not a requirement for Java itself, which is installed in SYS$COMMON. Many applications use file names that work on an ODS-2 disk, but it is something to be aware of if you get applications from other operating environments.

- UNIX doesn't have the concept of relative or indexed files, and everything is in what we would call STREAM_LF format. Java programs assume that as an access method, so you have to be careful to do an RMS CONVERT to get the data files into that format.

- UNIX has fairly loose process quotas. Sometimes this surprises people who are running Java programs on OpenVMS for the first time, when they don't perform as you expect them to. Check your process quotas to see if you are artificially limiting the performance, and observe what tuning changes you need to make. See the OpenVMS Java documentation for more details.

- The final restriction is on the hardware. Java runs on OpenVMS Alpha today, and will run on OpenVMS Itanium when that ships. Java will not run on OpenVMS VAX.

You can deploy Java on OpenVMS, but you can develop it anywhere. So take advantage of all of the development being done in other parts of your company, or even on the Internet, and compile them on the other platforms and deploy them on OpenVMS, just like any other platform.

Data Integration

Have you ever gone to a form on the web and started filling in the blank fields, and noticed that some of your information popped up to help you fill in the information? For example, your first name, your last name, your phone number, your FAX number, your street address, your city, your state, your ZIP code? How did the form know to fill in the right information at the right spot? This was probably done with eXtensible Markup Language (XML), which is a markup language for documents containing structured information.

Structured information contains both content (words, pictures) and some indication of what role that content plays. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

XML is similar to HTML in that they both separate form from content. But in HTML, both the tag semantics and the tag set are fixed. <h1> is always a first level heading, <b> is always bold, etc.

XML specifies neither semantics nor a tag set. XML is a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there's no predefined tag set, there can't be any preconceived semantics. So all tags are defined by the creator of the tag, and given meaning and usage by that creator.

As an example, both HTML and XML can show information that looks to us like a date.

- <b>5 May 2003</b>
- <ChangeDate>5 May 2003</>

The HTML document doesn't know anything about that information except that it should be displayed in bold at that point on the screen. The XML document defines something more about it: it is the change date. Elsewhere the "style sheet" has defined all sorts of attributes about this tag, including what format it is in (so you can display dates differently in different parts of the world), how to display it (so it will always be in bold) and what methods can be applied to it.

So it is not just text, it is actually information.

In our earlier example with the web form, at some point you filled in a form that had tags like <FirstName>, <LastName>, and so on, and your information was associated with those tags. The next time another web page asked you to fill in the form, the style sheet of that form used the standard definitions for those fields, and automatically picked up the information from the first form. It doesn't matter that the first form was running Microsoft IIS on a Windows platform and the second form was running BEA WebLogics on an OpenVMS platform -- XML provided the common definitions between the two forms.

All of the semantics of an XML document are defined either by the applications that process them or by style sheets. Style sheets define the attributes; that is, the format and the functionality of the tag. For example <FirstName> might be defined as text, specifies that there is only one token, and that it is a required value in this form.

You can create new definitions for data that is specific to your application as you need them. You might also consider looking into some of the industry-standard style sheets to use their definitions of common data. Not only is it easier on you, but your forms will integrate and cooperate with the forms of all other forms that use the same style sheet. This again reinforces the point that OpenVMS is just another platform as far as Internet technologies are concerned.

Now that we have common data definitions with all of the other platforms, we need to provide access to the data that is on our OpenVMS systems. We have seen how JDBC and XML provide the calling standards to let other systems look at RMS or Rdb or even Oracle 7/8/9i data on OpenVMS, but a calling standard isn't a running program. Attunity Connect provides that running program. Attunity Connect provides a universal integration solution across a wide range of enterprise systems consisting of varied application and data technologies on legacy platforms. Applications on other systems can simply access the data, without noticing that the data is on an OpenVMS system.

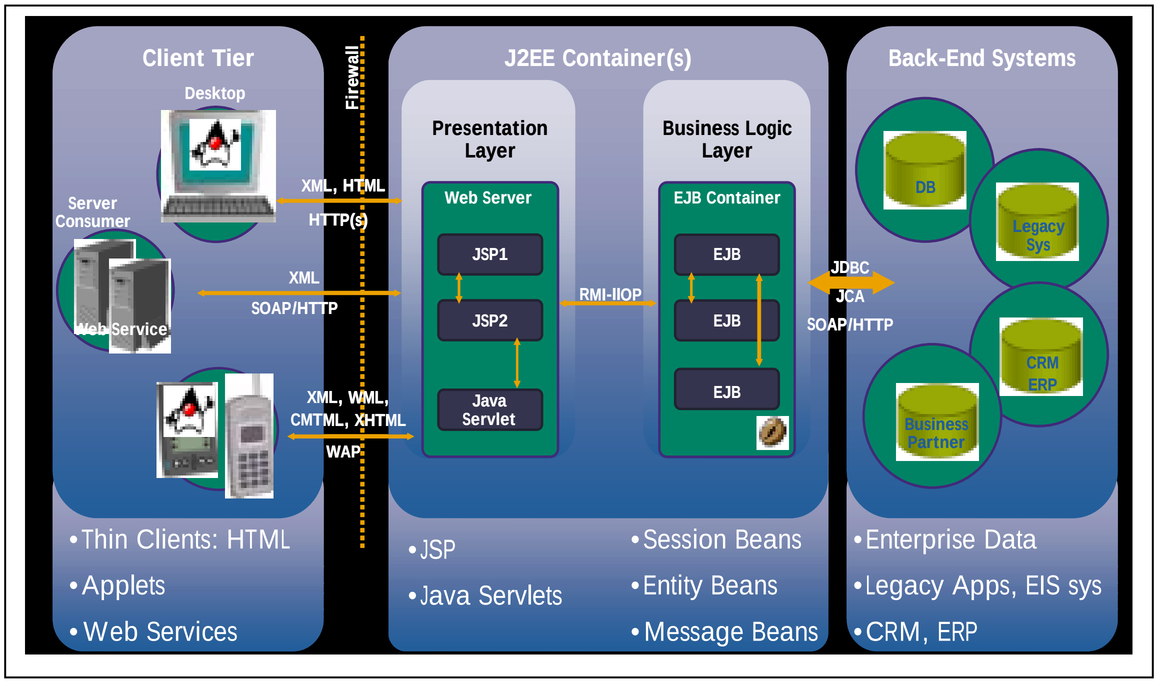
These systems consist of data and application resources that become available to the client application through Attunity Connect. A single Attunity Connect installation integrates applications as well as data sources on the same machine. Attunity Connect provides a universal integration solution both vertically from clients to servers and also horizontally across servers.

J2EE integration is made easy via J2EE Connector Architecture (JCA) and JDBC standards. XML can be also used directly.

What this means is that any client can directly access any data on your OpenVMS system. Java programs using JDBC, 3GL programs using ODBC, or Excel spreadsheets using XML, can just read and write data on your OpenVMS systems, without any special programming on their part.

This frequently makes it easier on you. If you give someone access to your data in their Excel spreadsheet, then they think it is up to them to write the fancy reporting programs and data analysis tools that they need, and not your job to do that for them. So you have shared the data, and they do the work. Sounds better than what is happening now, doesn't it?

So let's put it all together.



Starting from the right-hand side, the back-end systems, we find our enterprise data and many legacy applications. We access the data via JDBC and Attunity Connect, and specify the format and use of the data via XML. (We will talk about SOAP later.)

The new applications are written in Java so it doesn't matter where they run, so let's run them on OpenVMS. We create the business logic layer inside an Enterprise Java Beans container, which communicates to all of the EJB libraries that we imported from the Internet and other places, to give us the rich functionality our users expect. We communicate with the presentation layer running on a web server, again running on OpenVMS, running Java Server Pages and Java servlets. The two components can run one the same or different systems, running the same or different operating systems, and communicate via the Java Remote Method Invocation (JRMI) and the Internet Inter-Orb Protocol (IIOP), which again we will talk about in a few minutes. The session, entity, and message beans are simply data that exists for the life of that object: session beans exist while a session exists between the client and the server, entity beans exist while the object that created them exists, and message beans exist for the life of the message.

The client side is anything we want it to be. Windows clients, Linux clients, PocketPC or Palm clients, OpenVMS workstations, iPaqs, web phones – anything that can talk to a web server and display information. We communicate with this level through HTML, XML, Wireless Access Protocol (WAP), and so on.

The message to keep in mind here is that OpenVMS simply fits into each of the layers, and all of the software needed to run each layer exists on OpenVMS.

Application Integration

For applications to begin working with each other, they have to communicate in some way. The way applications communicate is via messaging. The generic term for this set of routines is Message Oriented Middleware (MOM). Middleware is just a fancy name for programs and APIs that allow one set of programs to interact with another set of programs, whether or not they happen to be on the same system or the same operating environment. You may already be familiar with some message oriented middleware, such as the Application Control Management System (ACMS) or the Reliable Transaction Router (RTR).

Message Oriented Middleware is, as the name implies, middleware focused on getting messages from here to there. But there are some difficulties with the wide varieties of “here” and “there” that exist, including distance between systems, transmission protocols from mailboxes to shared files to UNIX pipes to local area networks to wide area networks, and so on. In addition, networks sometimes aren’t as reliable as we need them to be, and some computer systems count the bits in a different order than other computer systems count them (some are big-endian and some are little-endian).

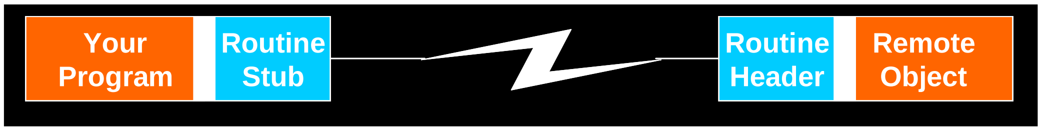
MOM takes care of all of this, by having a set of code on each system, including OpenVMS, which simply accepts messages from somewhere else, performs all of the right transformations on them, and passes them to the calling program. It does the same thing in reverse when one of your programs wants to send a message to some other system. Notice that MOM is totally uninterested in what the messages contain.

Examples of MOM include the previously mentioned RTR, as well as BEA MessageQ, IBM MQSeries, SpiritWave and SpiritJMQ from SpiritSoft (SpiritWave is the general purpose MOM while SpiritJMQ is focused on Java), BEA JMS and Tibco ActiveEnterprise.

Every one of these products runs native on OpenVMS, and simply communicates to a similar messaging system on any other platform. By adding calls to these routines to your existing applications, you can cooperate with applications on other systems, wherever they are.

The other set of tools for application integration is a higher level function, where the communications and messaging are taken for granted, and probably uses one of the tools we just talked about. This is the world of objects.

Objects are very similar to programming APIs, where you have a name of a function, a set of parameters to that function, and a set of return values from the function, all with datatypes carefully specified. The difference between objects and the programming APIs you are used to (such as calling RMS functions or SYS$QIO), is that these objects are not linked into your program. What is linked into your program are stubs, which act like the routine API that your program expects, but don't directly call code in your process address space. Stubs, when invoked by your program, use the MOM to call the routine somewhere else in the world, dynamically at runtime. The MOM passes the parameters, performs any transformation necessary, and then the remote object executes. The MOM then takes care of passing any data and return value back to the calling system, and your program never noticed that anything happened.



There are two forms of objects that currently exist in the industry. The first was created by an industry consortium, called the Common Object Request Broker Architecture (CORBA). They defined how to define stubs, message passing, return values, and so on, and then multiple companies implemented CORBA-compliant code on many platforms. JRMI Internet Inter-Orb Protocol (IIOP) is an example of CORBA-compliant code for the Java world, with IONA Orbix and 2ab Orb2 as two products that implement that specification.

The second form of object was created by Microsoft, and was known by various names including Component Object Model (COM), COM plus (COM+) and Distributed COM (DCOM). Today it is simply called COM. Due to the popularity of Microsoft platforms, all other platform vendors including OpenVMS implemented code to interact with COM objects.

More recently, Microsoft announced a development environment called .NET, running on the server platform called Windows 2003. This is an extension of the object model, and again all the platform vendors are implementing code to interact with .NET objects. Keep in mind that "interact with" doesn't mean that the target code runs on OpenVMS. It specifically means that your code running on OpenVMS communicates via COM with the code running on some other operating system, such as Windows 2003.

One problem with the MOM tools is that they often use protocols that can be filtered out by networking firewalls and other security devices. To get around this problem, people are using the Simple Object Access Protocol (SOAP). SOAP is a way of sending procedure calls from one system to another through Hyper Text Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), which is allowed through firewalls and other security devices, so it is quite transportable. It can be called from anywhere, including web pages and XML documents.

OpenVMS is fully compliant with all of these protocols, and objects running on OpenVMS can both call and be called by objects running on almost any other operating environment through this middleware. This means that you can begin using the web services that are currently running on other systems in your enterprise and throughout the Internet. In the next section we will talk about how to turn your existing code into objects, to make them into web services.

All of the APIs that you need to get your applications to use MOM are available on OpenVMS. However, many application developers today insist on using GUI-based development environments.

The HP Enterprise Toolkit -- OpenVMS Edition provides an interactive development environment based on Microsoft Visual C++ V6.0 and Visual Fortran V6.1. The HP Enterprise Toolkit -- OpenVMS Edition runs on Windows 95, Windows 98, Windows NT, and Windows 2000 systems and allows software developers to develop OpenVMS applications using an interactive PC environment. The developer can use C, C++, Fortran, COBOL, BASIC, Pascal, and Ada to write, compile, debug and tune applications in the familiar PC environment and run them in an OpenVMS computing environment.

The HP Enterprise Toolkit -- OpenVMS Edition adds several components to Visual Studio to provide additional software development functions such as: remote source file editing, remote compilation, linking, and building, remote find in files, remote debugging, source browsing, terminal emulation, and context-sensitive help. It also provides access to OpenVMS documentation, support for team programming, the sharing of project files among team members, user-defined configuration names, workspace-based source code control, support for using file shares to access remote project files, and built-in access to source code control on OpenVMS systems.

This simplifies software development with one environment for developing on multiple platforms, and provides a choice of tools and extends the industry-standard Microsoft development environment to OpenVMS.

In addition, the OpenVMS NetBeans team is currently developing a NetBeans plug-in that will allow distributed OpenVMS development to occur on any desktop that can run NetBeans, such as Linux, HP-UX, OpenVMS or Windows. The team is also creating a plug-in to support debugging of OpenVMS 3GL programs written in languages other than Java, such as C, C++ and FORTRAN, using the NetBeans debugger GUI and the Java Process Debug Architecture (JPDA) API.

HP plans to use this enhanced NetBeans environment to eventually replace the HP Enterprise Toolkit, to allow more flexibility in the development environments, and to provide a more standard development platform.

You have a choice: use the tools that you are used to today, or use the GUI-based development environments that are used for other platforms, either the Enterprise Toolkit from the Microsoft world, or the Java NetBeans environment for the Java world. Either way, you can use all of the message oriented middleware that allows you to communicate with other systems.

Legacy Application Integration

There are two ways to offer legacy OpenVMS applications to the rest of the enterprise as web services: you can add GUI front ends to existing applications with no change to the base application, or you can wrap the applications inside an object.

For applications written with DECforms, HP offers the DECforms Web Connector. This is a layered software product that runs on OpenVMS systems and provides transparent Web access to interactive applications, where DECforms was used to implement the forms-based user interface. The DECforms Web Connector lets you preserve large investments in DECforms-based user interfaces without requiring any programming or application changes. The DECforms Web Connector works by specifying a logical name at run-time that specifies the type of screen to present to the user: either the traditional DECforms screen on their current VT, or re-direct to a GUI platform such as Microsoft Windows or a web browser.

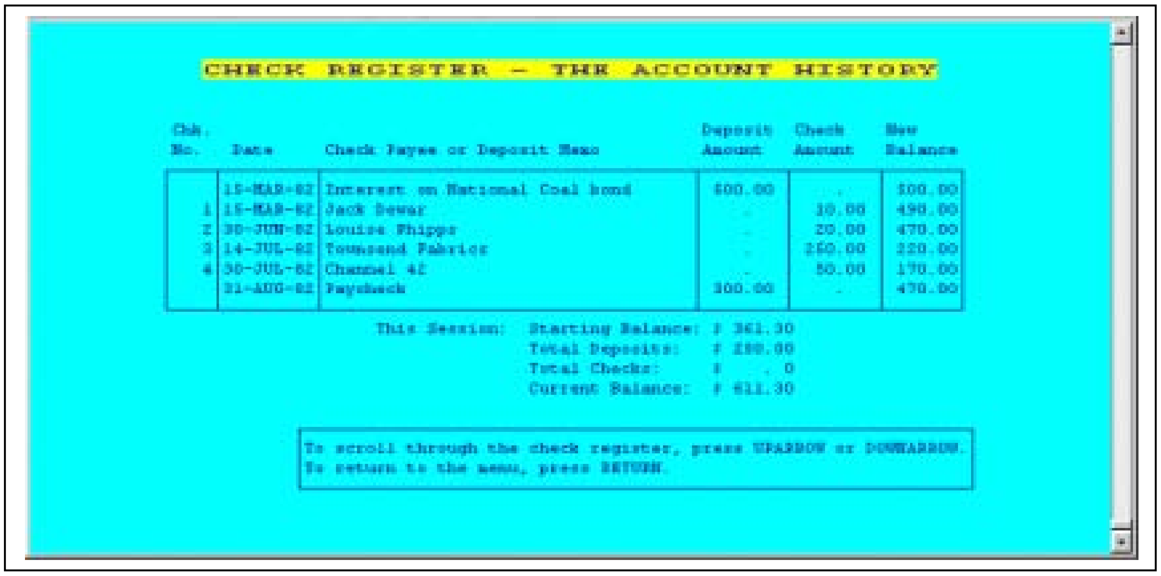
For the Application Control and Management System (ACMS) that runs on OpenVMS, and the cross-platform version (ACMS*xp*), HP offers the TP Web Connector, which web-enables business applications running on ACMS and ACMSxp for Windows NT transaction processing (TP) systems.

Using TP Web Connector, customers can create browser interfaces to any of these TP systems using a desktop tool that supports development with Automation, C-language, or Java.   TP Web Connector can also connect Windows NT (MTS) based systems to ACMS or ACMS*xp* for Windows systems. TP Web Connector supports integration of object modeling with critical business applications, all of the popular web server environments, and provides a high performance solution for web-enabling ACMS applications. The aim of this is not to migrate you off of ACMS, but in fact to do exactly the opposite: allow you to keep using ACMS but still give your application access to all of the other Internet technologies.

For a more general purpose solution, for applications that don't use DECforms or ACMS, HP works with Ericom to add new GUI front ends to existing "green screen" applications. The Host Publisher can even combine multiple applications into a single screen, for real enterprise application integration (EAI).

Here is an example of a forms-based VT-based application.

This is what the screen looks like on a VT terminal, just the way it has looked for years. Simple, clean, easy for us to work with, but not exactly the most modern look.



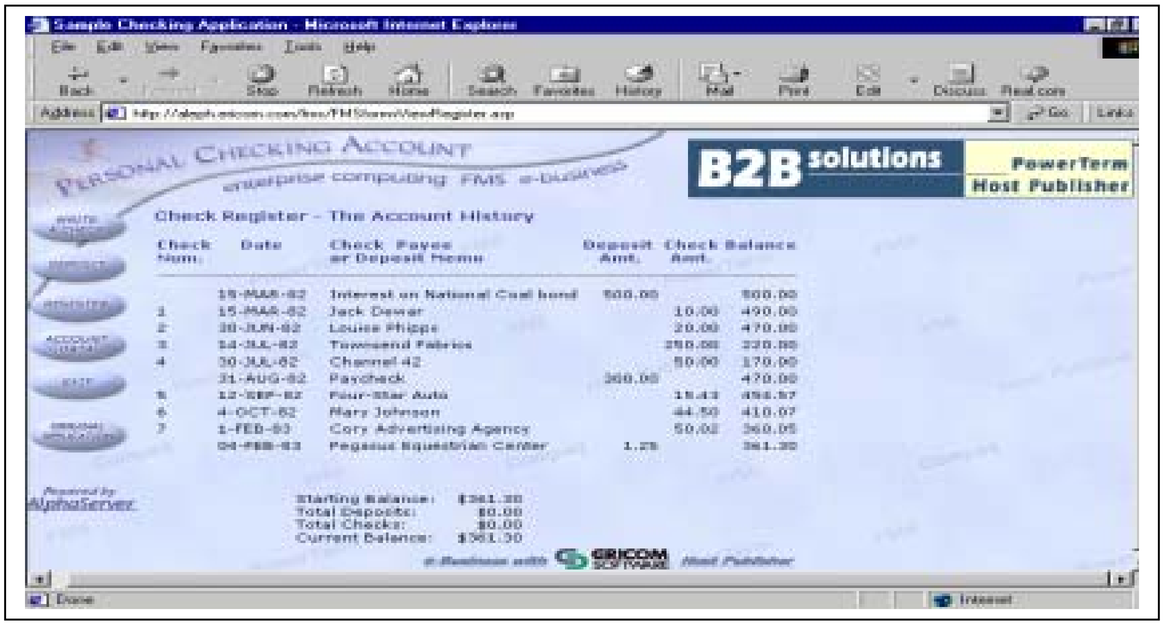
```
              CHECK  REGISTER  -  THE  ACCOUNT  HISTORY

 Chk.                                              Deposit   Check     New
 No.   Date       Check Payee or Deposit Memo      Amount    Amount    Balance
     15-MAR-82  Interest on National Coal bond      500.00       .      500.00
   1 15-MAR-82  Jack Dewar                              .      10.00    490.00
   2 30-JUN-82  Louise Phipps                           .      20.00    470.00
   3 14-JUL-82  Townsend Fabrics                        .     250.00    220.00
   4 30-JUL-82  Channel 42                              .      50.00    170.00
     31-AUG-82  Paycheck                            300.00       .      470.00

                        This Session:  Starting Balance: $ 361.30
                                       Total Deposits:   $ 250.00
                                       Total Checks:     $    . 0
                                       Current Balance:  $ 611.30

          To scroll through the check register, press UPARROW or DOWNARROW.
          To return to the menu, press RETURN.
```

And here is the way the same application looks with a new front end. Same data, same program, no re-compile, but something that people who have grown up on Windows or Macintosh screens are familiar with.


Sample Checking Application - Microsoft Internet Explorer
PERSONAL CHECKING ACCOUNT
enterprise computing FMS e-business
B2B solutions
PowerTerm Host Publisher
Powered by AlphaServer
e-Business with ERICOM SOFTWARE Host Publisher

Check Register - The Account History

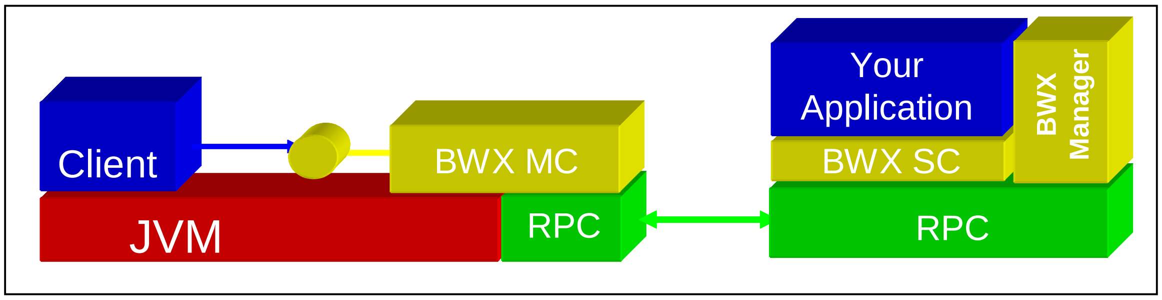
Check Num.	Date	Check Payee or Deposit Memo	Deposit Amt.	Check Amt.	Balance
	15-MAR-02	Interest on National Coal bond	500.00		500.00
1	15-MAR-02	Jack Dewar		10.00	490.00
2	30-JUN-02	Louise Phipps		20.00	470.00
3	14-JUL-02	Townsend Fabrics		250.00	220.00
4	30-JUL-02	Channel 42		50.00	170.00
	31-AUG-02	Paycheck	300.00		470.00
5	12-SEP-02	Four-Star Auto		15.43	454.57
6	4-OCT-02	Mary Johnson		44.50	410.07
7	1-FEB-03	Cory Advertising Agency		50.02	360.05
	04-FEB-03	Pegasus Equestrian Center	1.25		361.30

Starting Balance:	$361.30
Total Deposits:	$0.00
Total Checks:	$0.00
Current Balance:	$361.30

The original screen continues to work just the way it always has.

The other alternative is to encapsulate the existing applications. BridgeWorks can take existing applications, or pieces of applications, and turn them into components that can be used by external programs as Java Beans or as COM objects. The way this works is that BridgeWorks uses the OpenVMS Calling Standard to encapsulate the existing code into a wrapper whose external interface is either an EJB, a light weight Java Bean, or a COM object. Any part of your program that can be called by the OpenVMS Calling Standard can then be used as a standard Web component. You can even encapsulate your command procedures written in DCL!

Here is a detailed view of what HP BridgeWorks generates for the developer.



Your application is encapsulated by BridgeWorks (here abbreviated BWX), hidden by the Server Component (SC) and managed by the BridgeWorks Manager. This is exposed to the world via a message oriented middleware interface called the Remote Procedure Call (RPC) interface. The RPC interface can be either an EJB, Java Bean, or COM object, which you specify when you set up the encapsulation.

Your application communicates with the client side using the standard MOM technologies described in the last section, using either message-based or object-based communications. On the client side is the Messaging Component, which connects the standard client (browser, etc.) to the BridgeWorks wrapper to the application.

The client can be running on top of a Java Virtual Machine, or not, as is appropriate for your client.

In summary, you have years, even decades, worth of investment in your applications, which you want to preserve and use. Sometimes it makes business sense to re-write the application: you have learned better ways of doing things, it is in a language that is no longer popular, the business requirements have changed, and so on. But sometimes the program is working so well you just need a better front end on it, or a different way to call it. In these cases, OpenVMS has the ability to change the front end to a modern GUI, or encapsulate the functionality of the application into components that other systems will see as standard Java Beans or COM objects.

You can keep your investment, while still cooperating with the Web interfaces in the rest of your company. This allows you to take your applications and offer them as web services to the enterprise.

Systems Integration

OpenVMS has a standard web browser, called the HP Secure Web Browser (CSWB). It is based on the Mozilla open-source project started in 1998 by Netscape Communications Corporation. The Mozilla web browser is designed for standards compliance, performance, and portability. This is an example of the advantages I have been talking about up to this point: there is a perfectly good open-source program available, and it just runs on OpenVMS.

The Secure Web Browser is licensed as part of the OpenVMS operating system license and provides a full-featured, customizable browser with integrated web-browsing and security; HTML document creation and editing; clients for mail and news; Secure Socket Layer (SSL) for security, and an Internet Relay Chat (IRC) client is available.

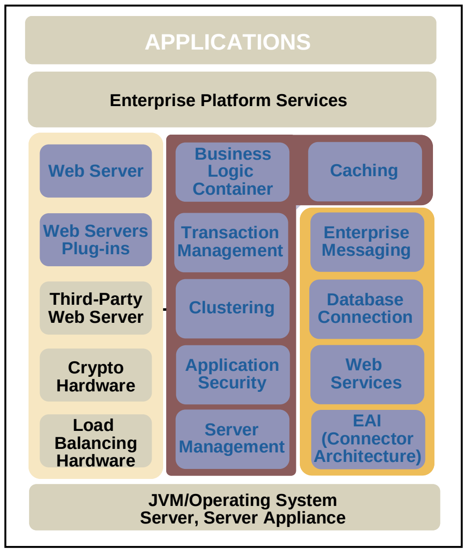
For web servers, we have the same scenario: the HP Secure Web Server (CSWS) is also based on industry standard open-source code, this time from the Apache Software Foundation. Apache is in use on millions of web servers around the world, on every major platform, and OpenVMS has the full functionality of Apache. And with all of the recent fuss over Internet viruses attacking web servers, keep in mind that Gartner Group recommends the use of Apache instead of Microsoft's IIS. So you get the enhanced reliability of Apache combined with the virus-proof nature of OpenVMS, for a very reliable and secure product.

CSWS, like the rest of Apache, is completely modular. Each function can be added to the server as you wish. Most people tend to add all of the available modules. Tomcat is the name of the Java module, with scripting languages like PHP Hypertext Processor (PHP, and yes, the acronym is recursive), Perl and Python modules freely available.

Web servers are all well and good, but all they do is present web pages, whether they are written in HTML or Perl or another language. They don't do everything we need, such as provide directory services, messaging, caching, and so son. For this we need a full application server.

Application servers integrate and coordinate all of the functionality that we have discussed. Your application simply assumes that all of this functionality is there, and the application server simply does it. You as a developer certainly could take pieces from everywhere and build your own application server, in the same way you could build your own operating system. The maintenance would be a nightmare, and there is no need because there are excellent application servers already available to you.

Let’s take a look at the components of an application server.



The first component is the web server described in the last section, along with its plug-ins like Java, Perl, and PHP. You can also plug in other components as you need them, including load balancing and high availability with OpenVMS Clusters (that is the “load balancing hardware” component in the lower left of the picture), as well as high security devices like cryptographic hardware and software.

The second component is the middleware for the middleware, if you will. Enterprise beans that you publish with common applications business logic are placed in an EJB container, available for use by any of your applications. The cluster manager handles transactions that are spread across multiple platforms using JTA, clustering of the application servers on different members in a cluster, caching of information in this server and between the application servers, and security of the transactions with JAAS.

The third component is the connection to higher level entities. The actual sending and receiving of messages via MOM is done here through the JMS interface, as is the JDBC connections to any databases opened by the application. Advertising the names of the web services offered by the applications running here is done with JNDI, as well as locating the names of services offered elsewhere and stored in the enterprise directory is done at this level. Finally, any connection to mainframe type applications such as SAP is handled here.

Quite a lot of work is done by the application server. In effect it is a mini-operating system, offering services to the applications that run on it. The advantage of application servers, however, is that they are common across many operating systems, so you can develop applications that run on one application server on one operating system, and they will just run on the same application server on another operating system.

The primary application server that HP recommends for OpenVMS is the BEA WebLogic Server. This is fully supported on OpenVMS, and brings the full functionality of BEA WLS to OpenVMS. Another excellent application server, which is not quite as well known, is the Xoology Concerto web server. Xoology has been running on OpenVMS for many years, and works very well.

One of the problems of MOM is that it creates a disconnect between the calling program and the called procedure. The called procedure is often an object, so there are various ways to invoke them, but one thing we have been ignoring is, how do we find them? How do we look them up, how do we tell which system they are on today, and how do we tell how they wish to be invoked?

The industry solution for this is the Lightweight Data Access Protocol (LDAP). LDAP describes how to access a directory of objects stored in a central location called an Enterprise Directory. In effect, the Enterprise Directory is a phone book of objects for your systems to use to find the services they need. You can store anything in there: people, phone numbers, e-mail addresses, other objects, a BridgeWorks wrapper around a DCL procedure, anything.

As you would expect, different vendors have different ideas on who should hold the data. Microsoft has implemented the Windows 2000 Active Directory, Novell has the Novell Directory Service, and there are two industry standards around this: X.500 and Kerberos. (Kerberos shipped with OpenVMS beginning with Version 7.3.) Finally, the Universal Description, Discovery and Integration (UDDI) protocol is a layer on top of XML and DNS, which acts as a building block to allow systems to quickly find and transact operations between disparate applications.

No matter which of these data holders you choose, OpenVMS can access it. OpenVMS can act as either a Directory System Agent (DSA), which holds the data and lets the Directory User Agents (DUA) on other systems access it, or can act as a DUA and can access the DSA on some other system.

So we can not only have your OpenVMS applications find all of the objects on the other systems, but we can advertise the objects that you will be developing on your OpenVMS systems.

Availability

OpenVMS considers all of the software described in this article as base functionality, and is included in OpenVMS. This software is available from the OpenVMS web site for download, or it is available as a CD bundled with every operating system kit. Further, the support for this software is included in the support contract for the base OpenVMS operating system license:

- The Secure Web Browser and Secure Web Server
- COM object technology
- All of Java, including the software development kit
- NetBeans
- SOAP
- The Attunity Connect JDBC drivers
- RTR
- The Enterprise Directory

- BridgeWorks
- XML for Java and C++

There is no additional license involved; it is all part of the base operating system. If you want to try any of this stuff out, simply grab it off the CD, or download it from our web site at http://h71000.www7.hp.com/ebusiness/technology.html.

I have been telling you how easy it is to bring software to OpenVMS, and how easy it is to take existing OpenVMS software and make it work with the Internet technologies. So you might very well ask if anyone has actually done this. Yes, many people have, and here are some examples:

- The GNU people are developing quite a lot of software that is in the UNIX style, but is specifically and deliberately not UNIX: their very name is Gnu's Not UNIX (which again is a recursive acronym like PHP). They developed a package called GNV, which stands for Gnu's Not VMS. This is a complete implementation of a UNIX shell, which we would call a command line interface, called BASH (the Bourne Again Shell). How many times have you run into a UNIX script that you couldn't run in DCL? GNV solves this problem quite nicely. Also part of GNV is a full UNIX C run-time library. The combination of the two of these makes it very easy to port full UNIX applications to OpenVMS without resorting to major surgery, including the ability of the application to spawn a UNIX script to perform some functions. UNIX system administrators and users really like this, and it is facilitating moving a lot of UNIX applications to OpenVMS.

- Another popular set of tools are GTK+ and libIDL. GTK+ is a development environment for creating graphical user interfaces, and libIDL is a development environment for creating CORBA interface definitions. These are part of many UNIX applications.

- There is a lot of secure information on your OpenVMS system and on other systems. Pretty Good Privacy (PGP) is a very popular package that encrypts information to an acceptable level: maybe not up to the requirements of the Defense community, but pretty good privacy. PGP is available in a package called GnuPG. The Secure Sockets Layer (SSL) is a popular method to secure network access between your OpenVMS applications and the outside world, or applications running on other systems that want to connect to the data on your OpenVMS systems. sTunnel is an implementation of SSL that runs at the network layer, so applications that currently don't understand security (such as telnet) can have secured communication.

- ZIP is an implementation of a very common file compression and packaging utility that runs on every system in the world.

- Finally, OpenVMS does not offer native support for writeable CD-ROMs. CD Record is an open source implementation of this functionality.

All of the above software is free, and available for download from http://h71000.www7.hp.com/opensource

Summary

You have a lot invested in your applications, your data, and in OpenVMS. You need to find a way to get the most return on that investment.

Every new technology goes through multiple stages:

- Phase 0 has a few small groups promoting something as the best thing since sliced bread.
- Phase 1 has a few more people adopt it as an alternative to the established standard.
- Phase 2 has people really understanding it, and beginning to see through the hype.
- Phase 3 has the product come back with more modest expectations and more solid engineering, and become a success.

Web services are somewhere between phases 1 and 2 right now, but we need to understand that the push toward inter-operating environments, regardless of platform, will not stop. It may or may not be Java, Windows 2003, XML or something else, but it is coming, and OpenVMS people need to understand this.

HP is partnering with the industry leaders to make this happen. Whether any given technology works out over the long term, these companies are going to be part of the cutting edge, and OpenVMS will be right there with them.

Acknowledgement

A great deal of this material was borrowed from seminars written and presented by John Apps and Mick Keyes. Without their assistance and review, this article would not have been possible. They are the real experts in this area.

For more information

All OpenVMS eBusiness technologies
  http://h71000.www7.hp.com/ebusiness/technology.html

OpenVMS Open Source Tools
  http://h71000.www7.hp.com/opensource/

OpenVMS Java
  http://h18012.www1.hp.com/java/download/index.html

OpenVMS CSWS (based on Apache)
  http://h71000.www7.hp.com/OpenVMS/products/ips/apache/csws.html

OpenVMS JServ
  http://h71000.www7.hp.com/OpenVMS/products/ips/apache/jserv.html

OpenVMS Perl
  http://h71000.www7.hp.com/OpenVMS/products/ips/apache/apache_perl.html

OpenVMS SOAP
  http://h71000.www7.hp.com/OpenVMS/products/ips/soap/soap.html

Apache Project
  http://www.apache.org/

Apache Java Project
  http://java.apache.org/

Apache JServ Project
http://java.apache.org/jserv/index.html

Java Servlets
http://jserv.javasoft.com/

Java Productss
http://www.java.sun.com/products

Web Services
http://www.webservices.org
http://www.ibm.com/developerworks/webservices/

Simple Object Access Protocol (SOAP)
http://msdn.microsoft.com/soap
http://www.develop.com/soap
http://www.soapware.org/
http://xml.apache.org/soap/faq

Books

Java Web Services by David A. Chappell & Tyler Jewell
Understanding Web Services by Eric Newcomer
HTML: The Definitive Guide, Chuck Musciano and Bill Kennedy
Apache Server Bible, Mohammed J. Kabir
Apache Server Unleashed, Rich Bowen and Ken Coar
Apache, The Definitive Guide, Ben Laurie and Peter Laurie
Writing Apache Modules with Perl and C: The Apache API and mod_perl, Lincoln Stein and Doug MacEachern
Professional Apache, Peter Wainwright
SSL & TLS, Designing and Building Secure Systems, Eric Rescorla
OpenVMS with Apache, OSU, and WASD, The Nonstop Webserver, Alan Winston

HTTP 1.1 specification, http://www.ietf.org/rfc/rfc2616.txt

HTML 4.01 specification, http://www.w3.org/TR/html4/