UNIX Code Migration Guide

UNIX Application Migration Guide

Chapter 15: Roadmap for Future Migrations

Larry Twork, Larry Mead, Bill Howison, JD Hicks, Lew Brodnax, Jim McMicking, Raju Sakthivel, David Holder, Jon Collins, Bill Loeffler
Microsoft Corporation

October 2002

Applies to:
- Microsoft® Windows®
- UNIX applications
- Microsoft .NET Framework
- XML

The *patterns & practices* team has decided to archive this content to allow us to streamline our latest content offerings on our main site and keep it focused on the newest, most relevant content. However, we will continue to make this content available because it is still of interest to some of our users.
We offer this content as-is, without warranty that it is still technically accurate as some of the material is undoubtedly outdated. Note that the content may contain URLs that were valid when originally published, but now link to sites or pages that no longer exist.

Summary: Chapter 15: Roadmap for Future Migrations examines the future for application development using Microsoft technologies. (12 printed pages)

Contents

Introduction

As technology advances, application migrations from UNIX to the Microsoft® Windows® operating system need to consider how to integrate and interoperate with new features and rising technologies on the Windows platform. This chapter introduces what to expect for current and future migrations with

regard to the following technologies:

- XML and Web services
- Microsoft .NET Framework
- Interix integration with the Microsoft .NET Framework

In addition, some recent developments in information technology might soon have an impact on a migration. These include the following, which are also introduced in this chapter:

- Microsoft Windows Server System
- High performance distributed computing

Migrating to XML and Web Services

The focus of this guide has been application migration, not application integration. However, application integration requirements are a major component of any application migration.

Recent developments such as eXtensible Markup Language (XML) and Simple Object Access Protocol (SOAP) have the potential to transform both the architecture and development of applications, and can have considerable impact on their migration.

XML, SOAP, and Interoperability

Most of the application migration scenarios discussed so far in this guide use some type of application programming interface (API) for interoperability. This may include one or more of the following API examples:

- Static libraries
- Dynamic libraries
- Shared memory sections
- Remote procedure calls (RPCs)
- COM/DCOM
- Common Object Request Broker Architecture (CORBA)

A major problem with any of these methods of integration is that applications require "compile time" knowledge to communicate with each other. For an RPC client to use a function exposed by an RPC server, the client must compile stub code generated by an interface definition language (IDL) compiler to actually make the call.

Interoperability and application integration would be much simpler if applications did not require compile time knowledge and if all potential clients and servers used common method invocation and marshaling rules. XML, SOAP and Web services provide such facilities.

XML

XML does not solve interoperability problems directly. However, it enables self-description of data items and data types, thus providing a basis for self-discovery of data definitions and common rules for marshaling. These concepts enable the standardization of cross-application interoperability. Because XML is a platform-independent technology, the value of XML-based services in an application migration is clear. Code developed to use XML can easily migrate from UNIX to Windows as long as

an XML parsing component is available on the target platform. This is certainly the case with Windows.

SOAP

The Simple Object Access Protocol (SOAP) builds on the capabilities of XML, applying them to method invocation and data marshaling. First, SOAP uses XML to describe the argument signature for methods, enabling the definition of signatures at run time instead of at compile time. Developers no longer have the burden of compiling IDL stubs into the application. Second, developers need not concern themselves with data marshaling because SOAP handles this. All applications need is a way to parse the SOAP message.

Shown below is a sample format for a SOAP message that uses an HTTP post method. This posts the SOAP call **SampleMethod** using an argument **arg1** with a value of 123456789 to the uniform resource identifier (URI) contained in the **POST** statement. The **MessageType** call indicates a SOAP request.

```
POST <URI of SOAP Method>
MethodName: SampleMethod
  InterfaceName: <Optional Name of Interface>
  MessageType: Call
  Content-Type: text/xml-SOAP
<SampleMethod>
    <arg1>123456789</arg1>
  </SampleMethod>
```

SOAP provides the basis for application interoperability on the Microsoft .NET platform. If a migrated application needs to access .NET services on Windows, the most important requirement is to build in SOAP support.

Web services

Building on the concepts of XML and SOAP, developers can also use Web services to bridge the gap between traditional UNIX distributed object models, such as CORBA, and XML on Windows. A Web service represents a way to expose SOAP calls by using HTTP. A Web server (such as Microsoft IIS) exposes the Web service by using a URL. Microsoft Internet Information Services (IIS) uses a special file extension (.asmx) to indicate a Web service.

Because SOAP and Web services invoke remote functions more flexibly, developers can wrap legacy distributed object models or method invocation methods either as Web services or as clients for Web services. Wrapping involves writing an additional layer of code that provides an interface between Web services and the legacy environment.

A Web service Description Language (WSDL) file enables access to the definitions of interfaces and arguments in a SOAP service in an XML format. If a client process needs to run a SOAP interface exposed on a Web server, the client can first request the WSDL XML file, and then use XML to discover the exposed interface and data types for the required arguments.

Leveraging XML in the Migration from UNIX

Although a migration to XML is probably out of scope for the base application migration from UNIX to Windows, XML can still play an important role. Using XML in migrated code can help bridge the legacy UNIX environment to the new XML-enabled tools on Windows.

This section looks at strategies to leverage XML-enabled tools on Windows by using two-tier UNIX applications. By its nature, data plays a primary role in this type of system.

As an example, let's look at the Microsoft Visual Studio® development system automation example from Chapter 7, Creating the Development Environment. This scenario presents a batch build development environment. The developer uses Visual Studio projects for debugging. The solution uses a Windows Scripting Host (WSH) script to automatically create a Visual Studio project based on project definitions contained in an XML file. The solution makes use of XML for the following reasons:

- Windows Scripting Host has built-in support for XML.
- The Microsoft Visual Basic® Scripting Edition (VBScript) script that it generates uses the Visual Studio Automation model and can also work with the XML DOM (document object model).

Note An *XML document*, in this chapter refers to XML contained in an XML schema.

In this example, the migrated batch makefiles act as the legacy data source. To extract the flat data file (makefile) and add this data to an XML document, a process is needed. For example, the process can parse the makefile for key words (such as CFLAGS, INCLUDE, LIB, and so on), then use WSH to generate XML based on the results. This can be automated by using Interix shell scripting to parse the makefile, and then using the output of the Interix script as input to a WSH script to generate the XML. Here, UNIX-style scripts are used at the front end to create XML. After the project definitions exist in XML, WSH can be used to create the Visual Studio projects.

The process of parsing the makefile and generating an XML project definition creates a "connector" or "adaptor" between XML and the non-XML environment. This is a common best practice for bridging XML data or other document types. Connector functions translate and map data between XML documents and external sources. This concept is discussed further later in this chapter.

Using XML and Web Services on Multiple Platforms

XML, SOAP, and Web services are not unique to the Windows platform. Most of the major software companies have adopted these technologies. Web services developed on Windows can be accessed from UNIX. Additionally, Web services written for UNIX can be accessed from Windows. In other words, it is simpler to migrate applications that use XML, SOAP, and Web services between the UNIX and Windows platforms.

For example, the Apache.org organization provides a SOAP toolkit for building Web services for the Apache Web Server on UNIX. This makes Web services available for legacy integration on UNIX. Its advantage is the use of SOAP for cross-platform integration. With SOAP, the Windows client need not implement legacy RPC calls or some other legacy UNIX network invocation method.

XML and Web Services Beyond the Firewall

The designers of XML and Web services created technologies that should function just as well in the extranet as they do in the intranet. However, the technologies create new challenges for virtual applications that cross the firewalls of corporations, such as:

- Security, including both authentication and authorization
- Cross-directory lookup, which requires a directory of directories
- Location service, that is, applications need to locate and attach to available Web services.

Security

Independent organizations need to agree on how to identify a valid user and how to determine that user's access rights. This requires an independent, trusted security authority that each organization can access for user credentials.

Microsoft Passport, for example, acts as a single sign-on authority for user credentials outside the corporate firewall. This provides the organizations in the virtual enterprise a common place to obtain a user's credentials without compromising private data about that user contained within an organization's firewall.

Directory lookup

Within the firewall, a corporation can implement an LDAP (Lightweight Directory Access Protocol) solution. To share some of this information with the users of other corporate directories, the corporation can use a metadirectory. A metadirectory is a way to look up common information that exists in multiple directories, which helps to solve the problem of resources located in another corporation's directories.

Microsoft offers a metadirectory solution called Microsoft Metadirectory Services (MMS).

Location service

After an enterprise has cross-firewall user credentials (by using Passport, for example) and can find users and resources in different directories (by using MMS), a developer targeting this environment still needs to find the appropriate Web services to perform the business functions. The Universal Description, Discovery, and Integration (UDDI) registry can perform this service. UDDI provides an application with a set of APIs to locate and bind to a particular type of service.

Migrating to the Microsoft .NET Framework

The Microsoft .NET Framework and the common language runtime (CLR) provide new features for creating powerful business applications efficiently and effectively. This section discusses some of the features of the Microsoft .NET Framework and the CLR, and how to leverage these features for code migrated from UNIX.

Introducing the Microsoft .NET Framework

The Microsoft .NET Framework is a set of software integration technologies based on industry standards such as XML, SOAP, and Web services. A development environment for .NET is Visual Studio .NET, a comprehensive tool set for building and integrating interoperable applications and XML Web services.

Microsoft .NET provides a number of features that developers can leverage during or following the migration of an application from UNIX. These include:

- Extensive support for scripting languages. This includes ECMA Script (Java Script) and VBScript. For example, Visual Studio .NET provides support for the integration of C# and Visual Basic .NET in the scripts for Active Server Pages (ASP.NET). This not only increases the performance of the ASP.NET server pages, but also allows for debugging with the same debug session as conventional languages.

- Multiple language support. The .NET Framework is language independent, enabling developers to use the same integrated development environment (IDE) and the same facilities whatever language they are writing in, including C, C++, and FORTRAN.
- Broad range of platforms. Using the Microsoft .NET Framework broadens the scope for platforms available to run code migrated from UNIX. A version of the .NET Framework (Compact .NET Framework) allows most managed code developed for Visual Studio .NET to run on Windows CE devices. This includes a broad range of devices ranging from embedded controls to the Pocket PC.

(For complete information on all the features available with Visual Studio .NET and the .NET Framework, see the MSDN Visual Studio .NET Developer Center.)

This section focuses on how Visual Studio .NET can be used to enhance and expand the functionality of an application migrated from UNIX.

The common language runtime (CLR) works with .NET to provide additional features, as described below.

Introducing the Common Language Runtime

The common language runtime (CLR) is a cross-platform run-time environment. It compiles multiple languages (including scripting languages) into a binary form known as the Intermediate Language (IL), which the CLR then runs. This allows optimizations for applications to target multiple platforms. Central to the use of the CLR is the principle of *managed* versus *unmanaged* code: Managed code runs under the control of the CLR; unmanaged code does not.

If cost reduction is a major business justification for the migration to Windows, using the CLR helps provide additional return on investment in the following ways:

- The CLR and Visual Studio .NET provide powerful tools to make it easier to access the same Windows features that justified the migration to Windows in the first place.
- The CLR provides system-level support for many of the features that a developer would have to otherwise develop.

The CLR provides the following features that are of particular interest for a migration:

- Common data types. This feature allows multiple languages to use and exchange data seamlessly. For example, C or C++ can exchange data with FORTRAN without special interface definitions in FORTRAN or C function mappings.
- Unified exception handling. This allows all the .NET Framework languages to use common exception handling routines.
- Cross-language garbage collection.
- Combined managed and unmanaged code in C and C++. This allows the same source program to integrate managed and unmanaged code.

Many languages are supported by the CLR, including C, C++, and C#, which are delivered with Visual Studio .NET. FORTRAN is available from third parties, such as Lahey-Fujitsu. This provides for multilanguage support from a single IDE. The multiplatform support combined with common data types and exception handling gives developers a powerful tool for building applications.

These features give the .NET Framework combined with Visual Studio .NET many of the integration

features with FORTRAN that otherwise require an integration strategy during a migration.

(For a complete list of CLR features, see the MSDN CDs or the MSDN Web site.)

Using .NET Framework and the CLR with Migrated Applications

The first major step to migrate legacy UNIX C, C++ and FORTRAN code to the Microsoft .NET Framework is to migrate the code to Windows. Features of the .NET Framework can then be integrated in the code migrated from UNIX.

Following the migration, there are four ways to add .NET and SOAP support to a migrated, unmanaged C or C++ application:

- Use the Microsoft Visual C++® development system SOAP toolkit. Managed code was not available with Visual Studio 6.0; therefore it bundled the SOAP toolkit. This toolkit allows the developer a convenient way to include SOAP support in an application without having to start from scratch with a SOAP parser. This same approach can be used with Visual Studio 6.0 or Visual Studio .NET. For documentation on the SOAP toolkit, see SOAP Toolkit.
- Use CLR **p-invoke** to access migrated unmanaged code from managed code. This assumes the unmanaged source is migrated as a DLL. Additionally, unmanaged code can be mixed with managed code by using C++ in Visual Studio .NET.
- Add a COM interface to unmanaged code, which can then interoperate with managed code by using the COM Interop feature of the Microsoft .NET Framework.
- Use Visual Studio .NET C++. By using the Microsoft .NET Framework native support for SOAP in managed code languages, the same C++ program can use both managed and unmanaged code. Source can migrate "as is," and then the developer can add functionality for SOAP by using managed code in the same project.

For unmanaged FORTRAN, the best strategy is to create a C or C++ library for the unmanaged FORTRAN to call. Then use one of the strategies listed above for integration with C or C++.

Accessing .NET Framework from Interix

Code and scripts ported to the Interix environment within Windows fall into the unmanaged code category of the Microsoft .NET Framework. However, it is still possible to provide interoperability for this application under .NET. There are two potential ways of doing this:

- Encapsulate as COM components Interix Applications written in C, C++, VBScript, Jscript or even the Korn and C shells.
- Run a program based on the Microsoft Win32® application-programming interface from a shell script within Interix.

The following sections discuss these options.

Encapsulation Using COM

To access Interix-based application functionality from a Win32-based application, define and develop a COM wrapper to provide Interix application functionality as a set of COM interfaces. The Win32-based application can call these interfaces to access services from the Interix application.

For example, consider an application that obtains its input from command-line options and provides its output to standard output (STDOUT). A COM interface can use IDL to pass the standard output information to a Win32-based application (such as a C++ application, Visual Basic application, or Microsoft Excel spreadsheet) or script (such as VBScript). The Win32 client makes a call to the Interix application COM DLL wrapper. The DLL invokes Posix.exe (and therefore, the Interix subsystem) to run the Interix application with the command-line options. The COM DLL captures and interprets the output and passes it back to the Win32-based application through the COM interface.

To build the application DLL library is by encapsulating the application

1. Define the COM interface by using IDL. Because the COM object needs to be usable from scripting applications such as VBScript and JavaScript, the interface needs to support ActiveX automation; that is, make use of **IDispatch::Invoke** as well as the custom interface.
2. Implement the interface in the DLL. Invoke Posix.exe to run the Interix application, capture the standard output, and then pass that data to the caller of the interface.

   Make sure the application command line is correct for Posix.exe. For example, it might require careful consideration of command-line quoting.

3. Build the DLL.

For more information about encapsulation of Interix applications in COM objects, see the Services for UNIX 3.0 Technical Note, "Interix and COM."

After a COM interface has been added to the unmanaged code—in this case, the UNIX application ported to Interix—it can then interoperate with the .NET managed code by using the COM Interop feature of the Microsoft .NET Framework.

Chaining Commands and Scripts from Web Services

In the Interix environment, Win32-based programs can be run from the Korn and C shells by using the SFU **runwin32** command. The Interix environment also includes shell scripts to invoke the standard Windows command-line programs, Cmd.exe "built-ins" are provided in the directory /usr/contrib/win32/bin.

Visual Studio .NET and the Microsoft .NET Framework also include support for ECMA Script (Java Script) and VBScript. The fact that the compiler compiles scripts into Intermediate Language does not change the way that Interix interacts with the Win32 environment to launch and chain commands and scripts.

Microsoft Windows Server System and Migration

Microsoft Windows Server System represents the next generation of server products for the Windows platform. The Windows Server System works hand-in-hand with the Microsoft .NET Framework to bring XML-enabled solutions to market.

This section covers key features supplied by Windows Server System that apply to other topics in this guide related to migrations from UNIX, including:

- How Windows Server System enables Enterprise Application Integration (EAI) based on XML

and Web services
- Standardization of XML data interchange by using XML schemas
- Using XML for data services with enterprise applications

EAI in a Web Services Environment

An application migration from UNIX to Windows needs to maintain application integration and coordination. The planning section of this book considered Enterprise Application Integration (EAI), which links disparate enterprise systems into a cohesive system for performing macro enterprise functions—at least, that has been the goal for EAI. Certain issues, such as the following, make this goal a challenge:

- Integration standards
- Data exchange
- Application coordination

XML and XML-based services (such as SOAP) address the first two items on the list, but help is still needed for application coordination. Microsoft helps solve this problem with Microsoft BizTalk® Server. BizTalk uses XML as its common language for data exchange and it adds coordination between applications.

BizTalk has three major functions:

- Integration definition includes data definitions and mappings. BizTalk Server uses XML schemas for data definitions and XML style sheets (XSL) for data mappings.
- Application messaging performs classic EAI functions including the following:
  - Component integration using COM
  - Web integration using HTTP
  - Queuing support
- Application orchestration enables logical relationships and dependencies to be created between independent applications.

Each of these elements works with the others in an integrated fashion. BizTalk Server orchestration provides the ability to receive or send output to and from the BizTalk messaging engine, the ability to process the data from the data definition and mapping functions, and the ability to create logical units of work across multiple applications. The logical units of work can include transaction semantics and support.

Web services can play a role in BizTalk Server systems by using the Web integration features or by using component integration. This provides system orchestration for applications that expose Web services.

XML DTD and Schemas

As discussed previously, the self-describing nature of XML introduces a new meaning to interoperability. However, just because an XML document self-describes does not make it usable. Applications still need to know what to look for.

This is where the value of XML schemas and document type definition (DTD) comes into play. By using a published schema, applications can set rules for the content of XML documents. A DTD also

describes the content of an XML document. DTDs predate schemas as an XML description mechanism.

The RosettaNet XML implementation represents a good example of XML in action. RosettaNet has developed a standard for data exchange and workflow for business transaction in the electronics industry.

By using a standard such as RosettaNet, partners in a supply chain can agree on the complex semantics of how to fulfill e-business transactions. Using the standard, applications can use Microsoft BizTalk Server to orchestrate the workflow needed to comply with the RosettaNet specifications. XML is a key technology that allows this to happen.

Although XML schemas and DTDs do not usually affect a migration from UNIX directly, the information can help an organization focus on what tools and technologies are needed after the migration to Windows.

XML-Based Data Services

Not only applications can be migrated from one platform to another. Data forms and structures can also be migrated. For example, developers can migrate legacy data structures to XML.

On the Windows platform, data typically takes two basic forms. The first is structured data, such as a relational database. The second is semistructured data. Microsoft Office documents, such as Microsoft Word or Excel files, represent semistructured data. XML has a place in data services for both structured and semistructured data. Unstructured data can be represented as objects or byte streams data elements in either semistructured data documents or structured databases.

Consider how XML interacts with a relational database. First, a relational database can store XML documents. However, XML can also be used for database queries. With Microsoft SQL Server™ 2000, Microsoft provides the ability to submit queries by using XML and to retrieve query results in an XML format. In this way, developers can encode the results of an SQL query directly in XML for use in a Web-based application. There is no need to translate the query results into an XML document.

Now, consider semistructured data. Because XML originated as an extensible data definition from a document encoding specification, SGML (Structured Graphics Markup Language), XML naturally lends itself to semistructured data. Therefore, the next generation of data storage for semistructured data on the Windows platform is XML-based. This enables document-based Windows servers, such as Microsoft Exchange and Microsoft SharePoint™ Portal Server, to use a common XML-based data store. An XML-based data store opens the possibility of using standards-based technologies such as WebDAV (Word Wide Web Distributed Authoring and Versioning) to access the data store.

Now consider developing a data query for a Web services environment. Because Web services operate efficiently in an environment where minimal state is maintained between the client and server, it is desirable to have "disconnected" results for database queries. The concept of disconnected query results is not new, but the .NET Framework version of ADO (Active Data Objects), ADO.NET, creates an XML document to package the results of a query. This provides a convenient way to browse the query results by using XML-enabled development tools.

High-Performance Distributed Computing

Techniques and technologies for high-performance distributed computing can impact a migration. High

performance computing can change the underlying architecture due to the requirements for compute power and network bandwidth. In addition, the potential distributed nature of high performance computing raises issues with testing and deployment, both of which now need to consider multiple computers rather than single computers.

With all the power now available on desktops and servers, the scientific and engineering community looks for a way to maximize the processing power of this commodity hardware and, potentially, the used cycles of these systems.

Two potential approaches are worthy of consideration:

- High-performance *cluster computing* refers to a technique where large numbers of the workstations or servers (or a combination of the two) work together to perform jobs that, until recently, only supercomputers could perform.
- *Grid computing* refers to using the network to link multiple computers so that tasks can be shared between them. Again, programmers and users can access significant computing power through the use of grid technologies. Furthermore, good use can be made of such CPU cycles, whereas before they would go to waste.

These two approaches are discussed further in the following sections.

Compute Clusters on Windows

High-performance compute clusters can include virtual computing jobs that carve up the computing task and distribute that task to available workstations and servers. The individual computing tasks can run independently or they might need to exchange information between tasks. Tasks that need to communicate and exchange information are often referred to as *grid computing*, which will be covered in more detail in the next section.

To create a compute cluster, the following types of services are needed:

- Resource monitoring
- Distributed job scheduling
- High-performance, low-latency message passing service
- Common scripting environment
- Remote login and execution
- Remote file transfer

Tools in the Windows environment available to perform these functions include the following:

- Interix shell scripts, Perl, and Windows Scripting Host (WSH) can support a common scripting language.
- Remote Shell (rsh), remote login (rlogin), and Telnet support remote login and execution.
- File transport protocol (ftp) and remote copy (rcp) support remote file transfers.
- Platform Computing, Inc. offers a Window-based version of Load Sharing Facility (LSF) to support resource monitoring and job scheduling.
- High-performance message passing is available by using the message-passing interface (MPI).

High-performance compute clusters are not unique to Windows. There are implementations on both UNIX and Linux. Because of this, a migration from UNIX or Linux to Windows might include

applications that need to target high-performance compute clusters.

For additional information about compute clusters, see the Cornell Theory Center (CTC) Web site.

Grid Computing

Grid computing is another way to exploit parallelism for computing-intensive problems by using cooperating computers on a network. With grid computing, applications split a task into smaller pieces that are sent to any number of computers. These computers send back answers to the controlling computer for merging into a complete solution.

For example, a corporation might use grid computing at night to take advantage of computers that otherwise run only screen saver programs.

A number of organizations offer screen savers that operate in this manner over the Internet to take advantage of an enormous amount of computing power on idle computers. The effect is to create the world's largest parallel computer. For more information about this, see:

- Global Grid Forum
- Grid Computing Info Centre (GRID Infoware)

Along with the concept of grid computing, a need was recognized for a standard way of communicating among the cooperating computers. Technologies evolved to pass messages between the cooperating computers. Two popular implementations of this are message passing interface (MPI) and parallel virtual machine (PVM) for both UNIX and Windows. PVM is older and has features that MPI does not, but MPI has become more popular in recent years and is the de facto standard for new projects.

PVM runs on Windows 9x, Windows NT, Windows 2000, most UNIX implementations, Linux and a host of other computers including supercomputers. Because the implementation is the same on all machines, porting from UNIX to Windows is straightforward after getting the distribution.

For more information about PVM and to download the release, see the official PVM site.

Two versions of MPI exist for Win32, a shareware version and MPI Pro, a commercial, supported version from MPI Software Technology. Because both of these and the UNIX versions implement the same standard API, porting the code is not complicated. Any changes necessary are because of other things that the application does and not because of differences in MPI implementation. These products both use the Win32 subsystem. There is currently no MPI implementation for Interix.

For more information about MPI, see the MPI Software Technology Web site and the Message Passing Interface Forum.

Send feedback to Microsoft