Chapter 1

Introduction to Windows DNA

If you are reading this book, you are probably a software developer who needs to understand Windows 2000 and Windows DNA. What you need is a solution that ties together all of the services and Component Object Model (COM) objects that you need to work with. And that is what you are going to learn. You are going to learn how to build distributed Windows 2000 applications. You are going to learn about Windows DNA and what it means.

Windows DNA is the Windows Distributed Internet Applications Architecture. Microsoft's marketing slogans have since changed, but Windows DNA is still what Microsoft originally described as a

> *blueprint that enables corporate developers and independent software vendors to design and build distributed business applications using technologies that are integral to the Windows platform.*

This means that Microsoft will be and is defining how to best build solutions using its technologies. The reason why Windows DNA and Windows 2000 solutions are synonymous is that Windows 2000 has all of the services that are needed in a full Windows DNA application.

I will be talking about the DNA model, but I will also be advocating the use of the technologies that best solve your client's problems. I have worked with Microsoft technologies for over ten years in a corporate setting, and the most important thing I have learned is that working with Microsoft technologies can produce good results, but you need to take things with a grain of salt.

The application of Windows DNA that I promote is what I found works best when applied to a client's problems. Many concepts that Windows DNA promotes are also available on other platforms, in the form of Java or Common Object Request Broker Architecture (CORBA). I am sure you are going to like it.

The Importance of Context

When I approach problems, I try to understand the context of the problem. The context includes and tries to make sense of the past and present of a current situation. When dealing with custom software, I often find that choices are made not only for technical reasons, but also for personal and other reasons. And the problem with personal reasons is that they are unpredictable. At times they are logical, and at times they are not logical. The decision is based on the context of the problem.

For this book, part of your context is to solve the problem using Windows DNA.

Let's establish the context of how Windows 2000 became the operating system it is. Consider this statement:

Windows 2000 is stable.

Did you laugh at that statement? I'll bet most of you did. We all have heard horror stories of the BSOD (blue screen of death) on Windows NT. You may think that Windows 2000 cannot ever be stable because of its track record. But if you inspect the past in detail and look at what Microsoft has attempted to achieve, you will see why certain things were done in certain ways. You will then understand how we came to Windows DNA.

Windows and Microsoft have been around for quite a few years. DOS was successful, but it was Windows 3 that started the Microsoft steamroller. Let's look at how Windows got to where it is now.

DOS—The First Era

When DOS first emerged, the context was the availability of an inexpensive computing device that could be programmed by a single person—it did not require the effort of large teams nor the budgets of multimillion-dollar corporations. DOS (disk operating system) ran on a device called the personal computer. Neither of these innovations were new. Apple developed its own personal computer and operating system, and Digital's CP/M was a type of disk operating system. What was new was that DOS could run on cheaper, generic hardware, and that DOS was cheap.

Windows 3.x—The Second Era

As the first era of personal computing grew, problems became apparent. Hardware was plentiful, but generic software with consistent drivers was not. Computer manufacturers made computers faster, and gave them bigger storage capacities and more RAM, but for software to take advantage of these new hardware innovations, the applications had to have custom drivers. Each application had its own drivers, which in most cases were incompatible with other applications.

Thus, the context was a growing amount of proprietary software that didn't interact well. The solution to this problem was Windows. Windows abstracted the concept of a device driver to be something that the operating system understood. This enabled people to buy software independently of hardware. Now anyone could run a piece of software, as long as there was a Windows driver for it. Hardware manufacturers only needed to write one driver to be used by all applications.

Windows 32-Bit—The Third Era

As the second era of computing grew, so did the problems associated with it. The second era of computing built on technological concepts introduced in the first era. For example, in the second era, applications used 16-bit processor commands, there was no multitasking, and there was no application fault protection.

Application faults caused the individual applications to crash, and often the entire Windows operating system would crash with the application.

This problem was a result of the driver model introduced in the second era. In the first era, every developer created their own systems. They had full control of the processor, hard disk, and screen. In the second era, applications had to share, but many people did not write applications or drivers that shared correctly. In fact, even Windows did not expose the interfaces properly, so the sharing caused faults. Writing systems that share is difficult because it requires a certain amount of trust that all other applications are doing their jobs properly.

The response to this context of bad sharing was to introduce multitasking and protected application space. The processor made the application believe that they were in the first era and that the application had exclusive control of the environment. The processor did this sharing at the chip level. This protected sharing made the operating system stable, as well as much more responsive and usable.

Windows DNA—The Fourth Era

So now we are at the fourth generation. Applications can now safely run in their protected space. What we still need is a way for applications to share a common plumbing without interfering with each other. The answer to this problem is the sharing of services. Sharing with other pieces in a system is extremely difficult to control because when a system has as widespread use as Windows, not everyone has the same idea of how things should function.

What has happened is that everyone developed their own version of various services. Some services were based on databases. Some services were based on the Web. Other services were based on transactions. And each of these services had its own interface. As users updated their shared versions of services, conflicts arose and systems started crashing. Users generally think that Windows is unstable because it is poorly written. In fact, Windows can be unstable because of the multitude of applications sharing the operating system and the compromises that are made. When applications share correctly, there is no problem.

To solve this problem, part of the Windows DNA strategy is to expose a set of common services and facilities that implement the "plumbing." The "plumbing" is the core services, such as transaction services, messaging services, and other components that can be reused in the various applications. This simplifies the concept of sharing, because there is only one set of services.

The goal of Windows DNA is to create a generic solution, which has these characteristics:

- Interacts with the user via a keyboard, screen, or mouse
- Potentially uses other input devices, such as touch screen, pen device, voice activation
- Connects to other machines using networking concepts
- Shares data with multiple machines in a reliable and consistent manner
- Manages itself and its environment easily

This type of solution calls for very different requirements than were considered in the Windows Application Programming Interface (API) era. Then,

developers focused on their own specific applications. The focus has changed now, because the computer has become a part of our daily lives and, as such, it must solve more tasks and be able to interact with other machines and environments. People want computers to work easily and simply, like toasters or radios.

The Windows DNA Strategy

In words that I have adapted from Microsoft literature, Windows DNA is definable as:

> The combination of an operating system that has a consistent set of reusable services with a logical design process to build applications.

There are three important aspects to this definition. The first is that it is an operating system. The operating system is a fundamental aspect of the computer system; it provides the basis of your entire framework. This operating system should be fast, stable, and scalable. The second aspect is that this operating system has a series of services and facilities. These services make it possible to do certain tasks, such as transaction processing or messaging. And finally, the operating system and services should be combined such that applications can be logically developed and take advantage of those services and facilities.

A Windows DNA solution is typically used by companies that are delivering applications to solve specific business problems. Microsoft already provides the plumbing. The size of the company using the solution does not matter—small companies and large companies generally have the same problems, the only difference being the scale.

The Guiding Principles of Windows DNA

The guiding principles of Windows DNA are as follows:

- Internet ready: Internet technology such as TCP/IP, HTTP, FTP, and so on, are integrated into the operating system. This makes it possible for applications to assume Internet availability.
- Shorter time to market: The built-in services and facilities allow developers to focus on writing the applications and not on the basic "plumbing," which enables faster development cycles.
- Interoperability: Open protocols and open standards make it possible for applications on various platforms to communicate easily and interoperate, so that a solution works with multiple vendors.
- Less complexity: Shared services and facilities are integrated within the operating system, and new versions are released with operating system upgrades. This avoids having incompatible versions of the same services installed by different application installers, making operating system releases and program installations simpler.
- Language independence: A strength of the Windows platform is that it is possible for third-party developers to choose their own development environments. This means developers can choose the language that will best solve the problem at hand.

- **Lowering the total cost of ownership: It should be simple to deploy, manage, and iterate the application over time. This can be achieved by simplifying installation and version-control processes.**

The Architecture of Windows DNA

Windows DNA is a tiered architecture. A tier is a logical separation of functionality from another piece of functionality. The tier may be an abstract concept, or it may be reflected in the actual components of a project.

Figure 1–1 shows the Windows DNA architecture, which is used to build multitier applications. There are three tiers: presentation, business logic, and data. Between the presentation tier and the business logic tier, there is a firewall. This architecture doesn't represent a physical architecture; it represents an abstract concept. Multiple physical tiers can exist within one of the abstract tiers.

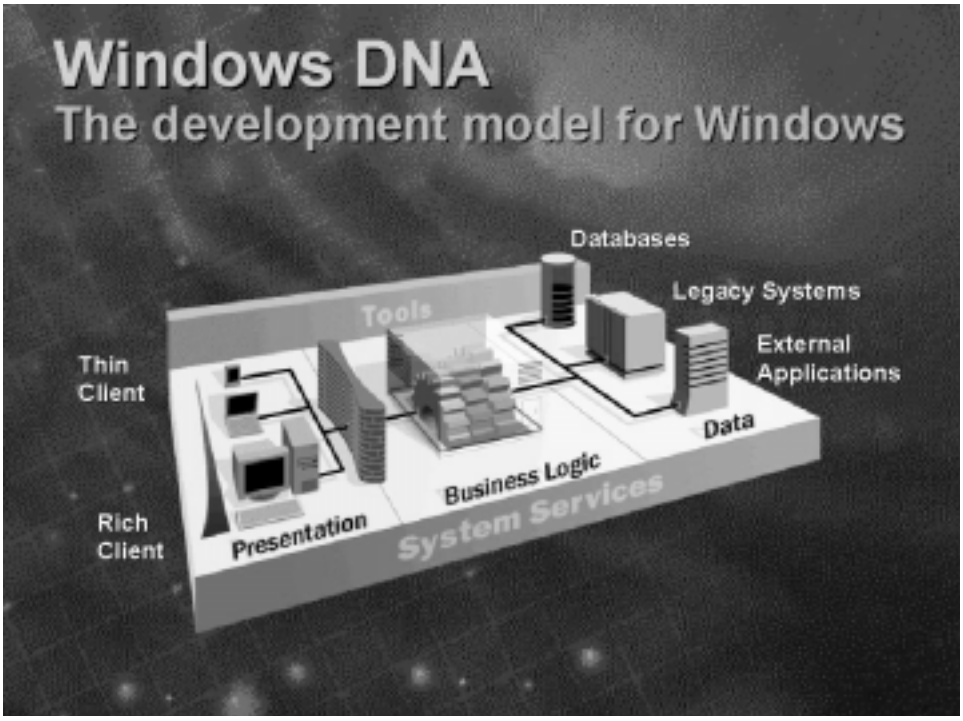

Figure 1–1: Windows DNA architecture

To make the entire architecture of a Windows DNA application work, the different tiers making up the application must be capable of communicating with each other without knowing their individual implementation details. This communication happens with a technology called COM.

COM Everywhere

The basis of COM is a binary standard that makes it possible for two pieces of code to exchange information. Figure 1–2 shows a vtable. The vtable is a series of function pointers to code implementations, and the vtable definition can exist on any platform—it is generated in a platform-neutral manner. Because the vtable is generic, it is called an interface. It simply defines specific intentions—the interface then points to implementations that actually perform the intended actions.

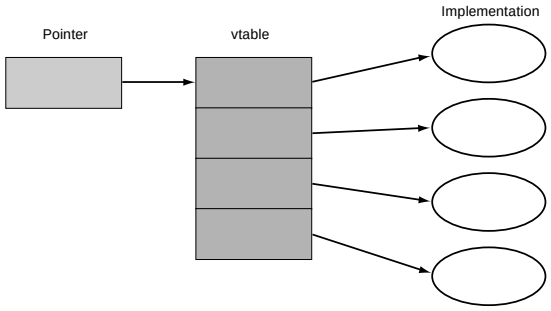

Figure 1–2: COM interface and implementation technique

The implementation has traditionally been binary and platform specific, but with the advent of Visual J++, this need not be the case. Visual J++ generates Java code, which can be executed on any platform. The only condition is that the COM run-time must exist on the destination platform.

When developing Windows DNA applications, you are likely to start by investigating the problem context. Then you define some intentions for the application. In other words, you define a series of interfaces. The implementations may implement intentions from one or more contexts. COM is used to query whether an implementation supports an intention in a specific context.

Windows 2000 in the context of Windows DNA does the same. The operating system exposes a number services, such as security, transactions, messaging, and hardware support, as a set of COM objects. This is in contrast to previous versions of Windows NT, where the majority of services were exposed as APIs.

So why COM? Besides the technical reasons, COM is currently the largest component strategy by far. By basing Windows DNA on COM, compatibility is improved. With COM it is possible to build software components that can be executed and deployed on any of the tiers. The COM run-time provides support for packaging, partitioning, and other multitier issues.

Presentation Tier

The presentation tier is the tier where the application interacts with the user. This is where the data is represented in a manner that we can understand. Currently this presentation is largely based on the keyboard, screen, and mouse,

but this will change in the future to provide better facilities for voice and gesture communication.

In the current version of Windows DNA, the focus on the presentation tier is to promote anywhere, anyhow, anytime access. This is made possible with Internet integration, including a browser and dial-up and networking facilities. Designing this tier is a challenge, because the types of clients vary dramatically. In the Windows DNA architecture, the client is not necessarily Windows based, but may include UNIX, Mac OS, and other operating systems.

To be capable of supporting various platforms, the presentation tier is graded from a rich client to a thin client. Please do not think of a thin client in terms of bytes to download. A thin client may be several hundred megabytes. The varying parameter is the amount of functionality that can be executed on the client side.

Figure 1–3 is an abstraction of how the presentation tier is built. On the right side, represented by a series of cogs, are the various technologies. Using these technologies, we can build two types of clients—EXE-based and page-based clients. Those types of clients can be further subdivided into four types, ranging from a thin client to a rich client.

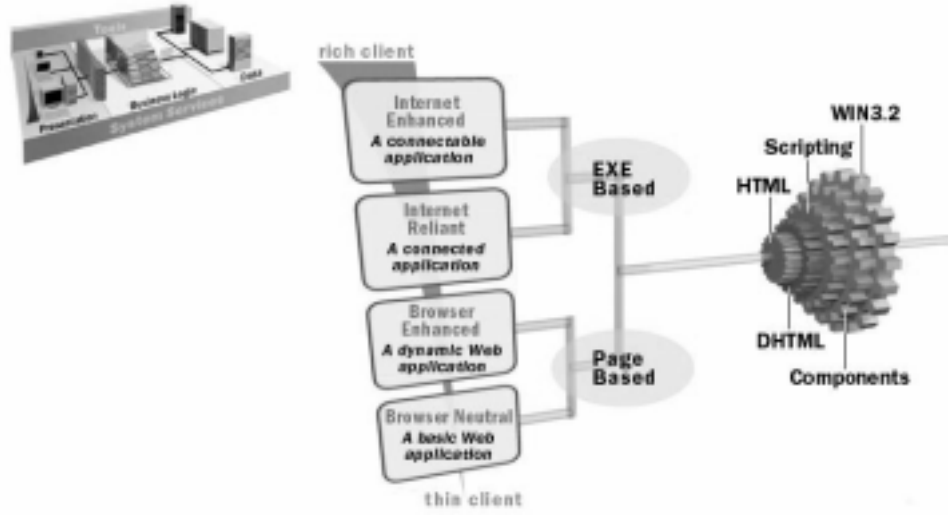

Figure 1–3: Presentation tier details

PAGE-BASED SOLUTIONS

The thin client is a type of page-based client. This means that the client machine is connected to the network and downloads content as it is required. The client application only displays and gathers information.

When a specific piece of content is downloaded, it is presented as static information. You can then fill out a few text boxes or set some checkboxes and submit it for processing. This type of functionality is very similar to batch-form processing in the mainframe days. The only difference is that the terminal can be anywhere on the Internet. Typically this type of client is a Web browser, and this

type of application is called a browser-neutral application. A static Web-browser approach ensures that your application has the broadest support.

The functionality on the client side is determined by the richness of the runtime on the client machine. There are simple Web browsers, or to get more functionality, a more sophisticated Web browser can be used. The type of solution is still the same, in that the application is downloaded. This is the first rich client shown in Figure 1–3, the browser-enhanced client. The difference between this client and the browser-neutral thin client is that some processing is performed on the client machine.

Using a browser-enhanced client makes the application more interactive, but the cost of this interaction is that fewer Web browsers can decipher the content. For example, it is possible to develop content that will function only on Internet Explorer or only on Netscape Navigator. Typical applications are Web browser-based intranet or extranet applications. Usually in this situation, it is possible to know which browser will be used.

EXE-BASED SOLUTIONS

The next two types of solutions require quite a bit of functionality on the client side; they require implementation of a fat-client application. These clients are based on executable applications installed and residing on the client machine, and they can be considered rich clients.

The first type of rich client is an Internet-reliant application that uses a mixture of the underlying operating system for functionality and the network for content. An example of this type of application is a data-warehousing application. When sifting through a data warehouse, you typically generate complex pivot tables and need a server to perform some of the work, because moving all the data to the client is not possible. To display the results in terms of a graph, however, you need a client that possesses a graphical subsystem. Hence, the application needs a dedicated network and rich-display functionality.

The last type of rich client is an Internet-enhanced application. This is similar to the previous client, except that it connects to the network whenever it needs to share information. Consider, for example, a word processor. Most of the functions that you want to do with it can be performed entirely on the client side. The network plays a secondary role in that it helps other word-processor users share documents.

This type of client is also breeding the new hybrid client application, which uses the Internet when it needs to, but does so invisibly. The user does not notice the difference. An example of this type of application is Microsoft Money. This application is installed on the client machine and can, for the most part, live in isolation. But when you want to retrieve your statements from the bank, you must go online. An embedded browser starts up and connects to your bank. From there it downloads your statements and displays them. During this entire process, the user never types a URL (Uniform Resource Locator) or gives any network commands aside from connecting to the Internet. In other words, the hybrid client made seamless use of the Internet.

THE TECHNOLOGIES

To understand what is going on behind the various client types, the core technologies need to be outlined:

- HTML: Hypertext markup language, the user-interface language of the Web. When only the term *HTML* is mentioned, typically it refers to Version 3.2 or, more recently, Version 4.0 of the HTML specifications. Both of these versions are plain-vanilla HTML, which supports form-based processing. This technology has the broadest reach among all technologies and is typically used in the thin client.
- DHTML: Dynamic HTML is the more advanced Web-browser user interface. Dynamic HTML differs from HTML in that it supports a full-fledged object model. Every part of the user interface can be dynamically addressed, removed, or added. This makes the user interface incredibly powerful. However, there are few browsers that support this feature.
- Scripting: Within the DHTML object model, a scripting interface is defined. It is considered a separate technology because with scripting it is possible to control any type of client. For example, Microsoft Excel can be scripted to perform certain tasks. The major benefit of scripting is that it allows applications to be customized without requiring complex programming skills.
- Components: Components are compiled pieces of code that represent some functionality. The DHTML and scripting services are components. Using COM, it is possible for these environments to communicate with each other.
- Win32: The Win32 API is a way of interacting with the Windows operating system. This technology is appropriate for those clients that need special graphics or high performance. The Win32 API is specific to Windows. It is not a similar programming model to HTML or DHTML.

Firewall

The firewall is not a tier, but it needs to be mentioned. Between the presentation tier and the business logic tier the firewall is erected. A firewall is a security wall that ensures the data on the inside of the firewall is not compromised.

In the general architecture, a firewall is only a service within the overall framework, but this is not the case with Windows DNA. Windows DNA is different from most operating systems in that it moves the security to the business logic tier.

There are several advantages to doing this. The first is that there is a consistent security-verification mechanism. Imagine your private network as a castle. Using current security mechanisms, a rider could ride into the castle anonymously. Then, depending on the service accessed, security verification is required. The problem is that as the rider moves through the castle, repeated ID checks are needed. As well, if one service is not as secure, an evil rider could cause destruction. Finally, with each service asking for identification it is not easy to keep a consistent security policy. If the rider is kicked out of one pub, all pubs in the castle should refuse to admit the rider. Without a central security authority, this does not happen.

By making the firewall a central security authority, anyone accessing any service is known. They are assigned a specific set of rights and privileges. If they abuse those rights, the rights can be revoked in an easy and consistent manner. In the end, it makes for a faster and simpler security framework.

Business Logic Tier

The business logic tier is the heart of your application. It processes and manipulates the data. The business process rules and workflow procedures handle this. This tier requires the most programming effort. This tier handles high-volume processing, transaction support, large-scale deployment, messaging, and possibly the Web interface. As you can imagine, the business logic tier is very complex.

If you are using Windows 2000 as a business-logic-tier server, there are three major services: COM+, Microsoft Message Queue (MSMQ), and Internet Information Server (IIS) (see Figure 1–4). There are other services, such as Microsoft Exchange and Microsoft Structured Query Language (SQL) Server, but they are not part of the standard Windows 2000 distribution. They are purchased separately.

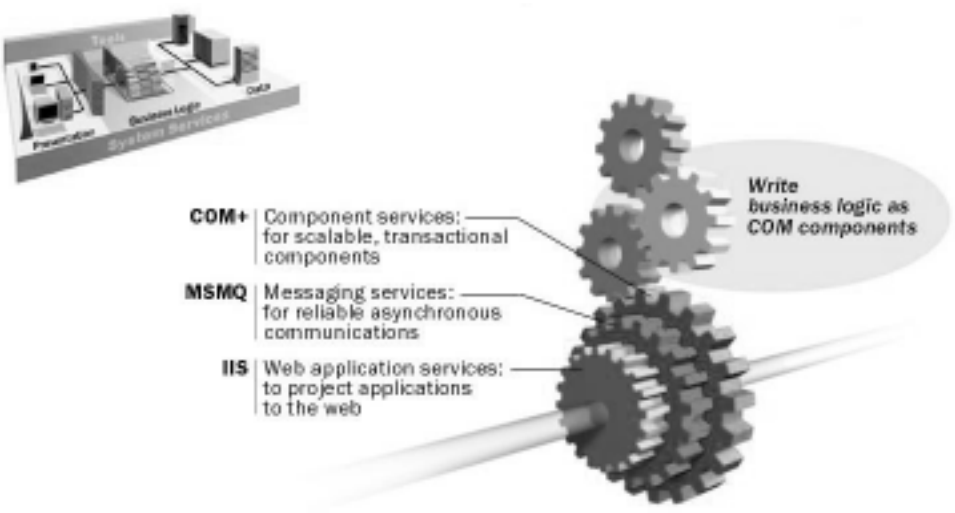

Figure 1–4: Architecture of the business logic tier

Each service performs a specific task. COM+ is responsible for the basic COM and transaction functionality. IIS is responsible for Internet services, such as SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), and HTTP (Hypertext Transfer Protocol). MSMQ is responsible for messaging. Each of these services may use other services to create new services. For example, queued components uses both COM+ and MSMQ to create a new service.

COM+

The next generation of COM is COM+. It extends COM by adding services that make COM more useful for the enterprise. COM only provides the component framework; COM+ adds transaction capabilities to COM.

COM+ and transaction services first came to life in SQL Server 6.5, which distributed a service called the DTC (Distributed Transaction Coordinator). This service was responsible for SQL Server transaction management. Then the DTC was re-released with a simpler wrapper called MTS (Microsoft Transaction Server). MTS made it possible to write COM objects that performed business processes and that could control transactions. COM+ takes the transaction aspect and ingrains it in the COM run-time and the operating system. MTS was also embedded in the operating system, but an MTS object could bypass MTS. COM+ closes these loopholes, and it is more consistent and elegant. From a programming point of view, COM+ permits all server-side COM objects to take advantage of transactions and make their applications more reliable.

The business logic tier contains by far the largest number of COM objects that provide specific services. This flexibility is required to build world-class enterprise applications that can receive, process, and send back data from the data tier. There are a number of enhancements that are based on COM and COM+, such as TIP (Transaction Internet Protocol) integration, enhanced security, in-memory database (IMDB), and dynamic load balancing, which are discussed in the following sections.

TIP Integration

COM+ uses Object Linking and Embedding (OLE) Transactions to manage the transaction-processing environment. However, that is only specific to the Windows 2000 platform. TIP makes it possible for COM objects to participate in other transaction-processing (TP) environments. These transactions can be managed by the TP system, which is different from the previous version in MTS. In that version, the transaction had to be managed by MTS and the DTC.

Enhanced Security

In the previous firewall discussion, it was explained that Windows 2000 enhanced the security model. The COM+ environment does quite a bit of that work. In COM+, security is either role-based security or process access permissions. Role-based security is a mechanism by which a role is assigned a set of permissions needed to perform a type of task, and then that role is assigned to the users who perform that task. Roles could be managers, employees, order processing, and so on. Roles make it simpler to assign different security privileges to users depending on their domain. For example, an accounting employee may have high access to the accounting processes, but low access to the manufacturing processes. An extension of role-based security in COM+ is the ability to apply security at the method level for COM objects.

Centralized Administration

In the previous edition of Windows NT, COM components were managed using the MTS Explorer. To manage any DCOM settings, the DCOMCNFG utility was used. In Windows 2000, both of these tools are replaced with the Component Services Explorer. It combines and extends the administration of the COM+ objects. Tasks include deployment, management, and monitoring of the COM+ application.

IMDB

One service that is new in Windows 2000 is IMDB. IMDB is an in-memory database that keeps tables in RAM. IMDB works with an OLE DB provider, which means SQL Server and ORACLE are usable. Many applications today require high performance, and most of the data used by those applications is read-only. An example would be the retrieval of a product catalog—this table changes, but very slowly. Most of the time it is read, and it could be cached. This is where IMDB serves its purpose. With IMDB, you can cache the product catalog tables so that it becomes a high-speed access table.

Queued Components

DCOM enables a COM object to be called between two different machines. DCOM is a synchronous protocol—when a call originates from one machine, the receiver must be available. If that receiver is not available, the call fails and any operation it is part of also fails. If that operation was executing within the context of a transaction, then the transaction will also fail. You must then implement a fail-and-retry mechanism.

In Windows 2000, an easier solution is to use queued components. Queued components make it possible to call another component on another machine asynchronously. Queued components use MSMQ as the underlying messaging architecture. Therefore, if the connection or call fails, it can retry until it succeeds.

Event Services

In Windows NT 4.*x*, it was not possible to easily connect events to components. There was a technology called COM connection points, but it was very limiting. COM+ events is a technology that permits loosely coupled events, where the event receiver may be active or not active. Using COM+ events, it is possible to make unicast or multicast calls—unicast means one-to-one and multicast means one-to-many. The receiver of the event connects to the sender of events using a subscription technique.

The publish and subscribe mechanism enables the publisher to inform the COM+ event service that it has information it wants to distribute. A receiver that is interested in this information indicates its interest to the COM+ event service using a subscription. Then, when the publisher has any information that changes, COM+ event services will propagate the changes to the various subscribers.

Dynamic Load Balancing

With the release of Windows 2000, it has become possible to dynamically load-balance a COM+ object. Dynamic load balancing is the technique of evenly distributing the processing load over a set of machines. Load balancing is a server-oriented process and it occurs transparently with respect to the client making the server call.

INTERNET SERVICES

There are various Internet services that are provided by the IIS.

- HTTP: This is the protocol of the Web, and it serves up Web pages or any content that the Web browser requests.
- FTP: This is the protocol used on the Internet to transfer files. It is specifically geared toward file transfer and is therefore more efficient for that task.
- SMTP: This is the protocol used to transfer mail messages from one machine to another. It can be used in conjunction with POP.
- Gopher: This protocol existed in previous editions of IIS, but has become obsolete, since Gopher was superceded by HTTP.

What makes IIS special with respect to other Internet servers is its integration with transactions. When a request is made to IIS, it is a transaction request. By using transactions as a basis for handling requests, all of the benefits of COM+ can be realized. This also makes IIS more flexible when it needs to be dynamically load-balanced or optimized.

To build content for IIS, it is possible to use ISAPI (Internet Server API) or, more commonly, ASP (Active Server Pages). ASP is a scripting environment, where logic is executed from scripts. The script language is neutral and totally dynamic. It is used in conjunction with the COM+ object to provide a full-featured business application. ASP can be used to generate DHTML or XML (Extensible Markup Language). While not part of Windows 2000, the process of creating ASP, DHTML, and XML applications can be handled with Microsoft Visual InterDev.

MESSAGING SERVICES

Asynchronous messaging is handled by MSMQ. Messaging is a totally different way of writing code. It assumes that the connection between the client and server does not exist, but that eventually the operation will be carried out. The eventuality model makes it possible to write applications that function over unreliable networks or in unreliable conditions.

MSMQ is an event-driven architecture. The data is pushed to the server and then the server must react to the event being called. The event-driven model is nothing new, because Windows itself is event driven. However, it does mean application programming must be different from the usual norm, since an immediate reply is not to be expected.

INTEROPERATING WITH EXISTING SYSTEMS, APPLICATIONS, AND DATA

Making Windows 2000 Internet-ready was one task. But another very big task is to prepare it for the integration of legacy systems. Legacy systems are legacy because they work and do the task well enough. Therefore, replacing these systems is not an option. Instead, these systems must be integrated into the overall architecture.

Windows 2000 provides options for integrating various legacy technologies:

- MSMQ Integration: Messaging integration is provided for MSMQ using various products. To connect from the Windows platform to the MQSeries Product, SNA Server can be used. To connect from other platforms (UNIX, MVS, OS/2, and so on) to MSMQ, FalconMQ client from Level8 can be used.
- COM Transaction Integrator: Using this interface, it is possible to extend CICS and IMS transactions (messaging and transaction products offered by IBM) as a series of COM objects. In the COM Transaction Integrator, there is a series of development tools and business logic that wraps the IBM mainframe transaction. The communication with the IBM transaction is handled by Windows SNA server.
- Universal Data Access: This layer is the database-access layer. By using native drivers it is possible to connect to resources that reside on the other platform's side.
- DCOM on UNIX and Mainframe: This is a technology that makes it possible to write COM objects on platforms other than Windows. Software AG provided the first port to Solaris. That porting has been extended to include the mainframe platform and currently extends to a full architecture called EntireX iNTegrator.
- TIP/XA-Transaction Integration: There exist several transaction protocol standards. TIP has already been discussed, and another standard is the XA-Protocol. The integration is provided through a resource dispenser/manager.

Data Tier

The last tier is the data tier. It is the resource-management tier. In contrast to the business logic tier, this tier does no processing of data. Instead, the task of managing the huge amount of data is defined. Databases and resources are getting bigger every day and the task of managing the data is becoming more and more difficult.

Some data is in spreadsheets, some in word processing documents, and yet more in relational databases. There are two ways to store this variety and amount of data. The first way is to create a storage repository that can handle all of these different data types. In current-day practice, this way of storing data uses blobs (binary large objects).

However, blobs only manage data as binary images. They do not ensure relations. For example, if a stored blob referenced another document, an invalid reference would not be caught. At the data tier, this invalid reference means nothing, but it is important at the business logic tier. It could cause a transaction or multiple transactions to be aborted.

One solution is to ensure that the business logic tier writes a correct blob. However, this adds extra programming steps that should be part of the data tier—because reference management is a data operation. To add reference management, an external module must be added to the data repository. This is not the optimal solution.

A better solution is to decide that each resource handles its own data format, but that the resource must adhere to a common object model. This makes it possible to manipulate, copy, and reference data in a format-neutral manner. This is what the Universal Data Access (UDA) strategy offers.

The UDA object model is called the OLE DB object model (see Figure 1–5). It is a low-level format that includes the specification of how transactions and rowsets are defined. The OLE DB object is very high speed and low level. This means that writing to OLE DB requires quite a bit of work. While it is not part of the Windows 2000 distribution, Visual C++ has a thin template layer on top called OLE DB Consumer templates. It simplifies OLE DB, without making it slower or less efficient.

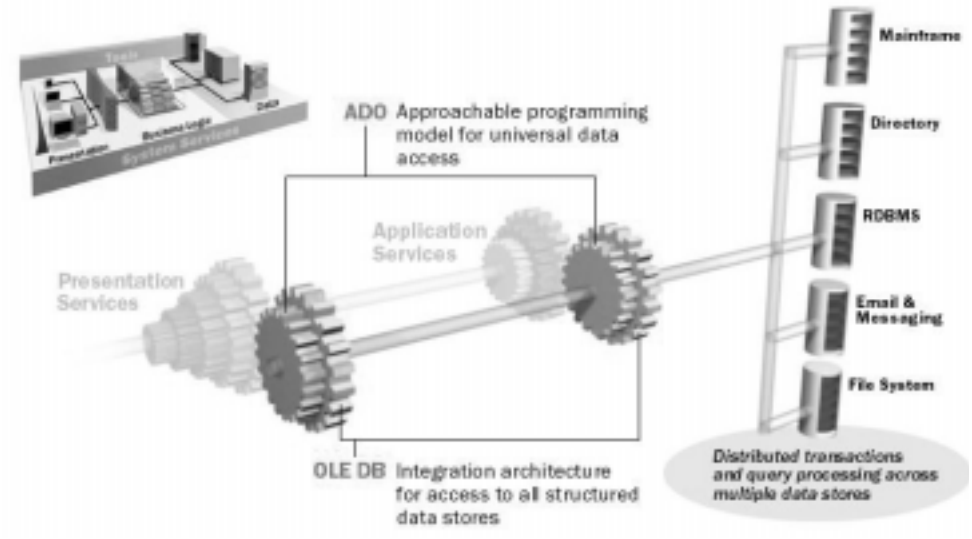

Figure 1–5: Universal data access (UDA)

The advantage of the OLE DB object model is that it is not limited by a particular shape or format. For example, in a relational mode, the only resultset possible is rectangular. It must consist of a number of rows and columns. OLE DB has a minimum number of COM interfaces that must be implemented, but beyond that, it is up to the OLE DB provider. There are no limits other than that it must be a COM object.

OLE DB is a very low-level technology and only compiler-based development environments, such as Visual C++ or Delphi, can use it. For the Visual Basic programmer or the script writer, a simpler object model called Active Data Objects

(ADO) is made available. ADO is not less powerful, as it does enable access to most of the features of OLE DB.

One question that must be answered is this: How does OLE DB expose the functionality of relational database blobs? The answer is by using COM objects. This is why ADO does not need to handle all the various data formats. It is the responsibility of the provider to generate the COM objects that will provide the interface between the data and the ADO calls.

Writing Applications Using Visual Studio

The strategy of Windows DNA is not focused only on the services and facilities in Windows 2000. It also involves building applications using these services and facilities. This is what this book focuses on.

To build applications, a set of tools must be used, and Microsoft Visual Studio is a suite that contains all of the necessary tools. There are various versions of this suite. The one that you are interested in is Visual Studio Enterprise Edition, which could also be called Visual Studio DNA Edition. This is because it contains all of the tools necessary to build a Windows DNA application.

One problem with Visual Studio is that it contains various programming languages. This causes friction because people tend to choose their favorite language(s). A strategy for implementing Windows DNA is to choose the best language for the job. This changes the debate from which language is best, to which is best when.

In this book, I have chosen a random approach based on whichever language felt best at the moment. Now before you judge me bonkers, remember that you are a Windows DNA engineer, and as such you need to understand the context and history of this decision.

The Context of a Random Language

A friend of mine is a consultant who builds vertical applications and customizes them according to the client. This is a classic Windows DNA application.

When he developed his applications, they had these characteristics:

- The applications had a full product development cycle (designing, testing, documentation, etc.)
- The applications had a full lifetime cycle (bug fixing, versioning)
- Each step of the product development cycle was carried out in the most efficient manner possible

My friend had some very hard decisions to make when it came to choosing a language for his applications. In the first version of his application, he decided to write in C++, because some legacy applications were present. However, he wanted the optimal return on his investment. His team did an internal study focused on the entire product development cycle, looking at various projects that used various languages.

What were his conclusions? That the language does not matter. What matters is how good your team is. Experienced developers made all the difference. In

the details of the study, though, there were some interesting findings to do with the specific languages.

Let's start with the initial productivity of the programmer. Visual Basic is an environment that makes it possible to quickly implement an application. Visual C++ was slower because it required some library and object development.

Another measure of productivity is how fast a programmer can learn the language. There, again, Visual Basic is superior, because learning it is simpler. Visual C++ is more difficult to learn and to program with.

Another measure of productivity is the number of bugs produced in the code written. This time Visual Basic fails miserably. For every bug in Visual C++, there were three bugs in Visual Basic.

So, which language is more productive? It would seem Visual Basic. Even though there are more bugs, fixing them is faster because the productivity is that much higher. Unfortunately, that answer is incorrect. The real answer is that Visual Basic is as difficult a language as Visual C++.

What tips the scale is the nature of the bugs in Visual Basic. In Visual Basic, the bugs are logical, whereas in Visual C++, the bugs are bad pointers or corrupted memory. Logical bugs are bad for the product development cycle because they require changes in documentation, design, and so on. All of these can require extra time and money. Bugs in Visual C++ are simpler because they generally only require changes to the code and not to the logic of the code.

Why is this the case? Visual C++ is a difficult language that requires a lot of learning. You are forced to ensure that you dot your *i*'s and cross your *t*'s. This means a beginner Visual C++ programmer has more experience than a beginner Visual Basic programmer. This experience ensures that the Visual C++ programmer has the ability to properly design classes and systems.

So what was my friend's decision in the end? Get the right programmer! When experienced Visual Basic programmers were used, the number of logical bugs dropped dramatically. This is why there is no right language for the job—there is only a right developer for the job. And this is why this book uses a variety of languages for the examples. Naturally, I do not use batch scripting to build logical components. But I do, at times, use Visual Basic, or Visual J++, or even Visual C++. But throughout this book, I always show you the correct technique of applying the technology. This is part of building Windows DNA applications using Windows 2000.

Programming Languages in Visual Studio

The basic edition of Visual Studio has five development language products.

VISUAL J++

Visual J++ is a development tool that is based on the Java language. Java is a language that takes source code and compiles it to an intermediary bytecode. The bytecode is then interpreted by a virtual machine. Using this approach it is possible to compile sources and then have them execute on any machine that has a Java virtual machine.

Sun Microsystems developed the original Java language and also defines Java's standard libraries. The Visual J++ product supports most Sun Java library calls, but is geared toward Windows development. This makes Visual J++ somewhat incompatible with the standard Java libraries. Also, at the time of this writing there is still an outstanding decision regarding Microsoft Java and Sun Java. This is causing Microsoft to delay the release of a new version of Visual J++.

Advantages

The Java language is simpler to use than C++, but it is more complicated than Visual Basic. It is a pure object-oriented language, which forces the developer to develop using objects. It does not understand the pointers that make C++ development very complex. The interface concept is integrated into the language, which makes component development a natural evolution. The Visual J++ development environment currently in the Visual Studio environment offers the most productive environment. It integrates documentation with the product.

Disadvantages

The biggest disadvantage is the speed and flexibility. Java can be viewed as being simple and yet expressive, or it can be viewed as complex and limiting. It really depends on whether you have a positive attitude about Java. The current implementation of Visual J++ does support the library set of Sun Java, but the productive aspects are not compatible with Sun Java. While cross-platform capabilities are one of Java's greatest features, this isn't well supported in the Visual J++ context.

VISUAL C++

This is a development tool that uses C++ as the development language. It is the development tool that is used most often for systems and software application development. This language has the capability to integrate C and COM, and there are no limitations.

Advantages

The C++ language is by far the most popular development language for systems or software applications. It supports all aspects of object-oriented development, while having the capability to integrate legacy C source code. It is a language that can do everything, and is a natural choice for building libraries, objects, and components. With an experienced developer, the code tends to be more reliable than other languages because of the capability to tie in debug libraries and objects. These objects can be used to focus in on run-time and compile-time inconsistencies.

Disadvantages

C++ may sound like a dream development language, but it does have its share of problems. C++ provides the ultimate development and design freedom and flexibility, but programming errors can produce disastrous results. It is also a difficult language to learn. With all this power, many C++ developers have their own development strategies, and these varying techniques can complicate team development.

VISUAL BASIC

This is one of the most popular development tools on the Windows platform. It is a language that is based on a Microsoft-specific version of Basic. The Visual Basic 6.0 edition supports object-oriented constructs, such as objects, but not inheritance. It masks the complexities of the operating system as a series of COM objects.

Advantages

The biggest advantage of Visual Basic is its popularity. It is very productive because components and applications are easily developed. The language and environment are simple and easy to learn.

Disadvantages

The biggest disadvantage of Visual Basic is its simplicity. It leads to design-and-code-as-you-go applications. While these applications will not have as many development errors as they would in Visual C++ (pointer errors), Visual Basic tends to produce more logical errors. This sort of error can be more expensive to fix, because logical errors require a redesign of the logic. Because Visual Basic tends to abstract the operating system and services as a series of COM objects, the distribution of a Visual Basic application is more challenging because of the size of the final application.

VISUAL FOXPRO

In the previous explanations of development environments, each of the tools represented a code-and-compile scenario. Visual FoxPro is different because it is a development environment integrated with a database environment. This makes developing database applications much simpler. It is object oriented and extremely fast when making database accesses.

Advantages

Visual FoxPro developers tend to be incredibly talented and to understand their development tool very well. This means FoxPro applications tend to be well designed and developed. FoxPro is an all-in-one environment that has full object-oriented capabilities.

Disadvantages

The developer pool, while constant, is smaller than that of the other tools. FoxPro is a classic client/server development tool. It is not the ideal component-development tool.

VISUAL INTERDEV

The last development tool in the Visual Studio development environment is the Dynamic HTML and ASP tool Visual InterDev. Visual InterDev does not attempt to be an all-in-one tool. It is a tool used to build Web applications and make use of components. While Dynamic HTML is a standard, Microsoft Internet Explorer is the only browser that supports it.

Advantages

Visual InterDev is the development tool for Dynamic HTML and ASP. It is well-suited for and makes it simple to build full-scale Web applications.

Disadvantages

Visual InterDev tends to overcomplicate something that could have been expressed more simply. While it does a good job of moving the classic programmer to a Web environment, for those developers who grew up with the Web, it is not rich enough or diverse enough. It can be used to build Web pages for Netscape Navigator or other browsers, but client-side scripting tends to be more suited for the Microsoft Internet Explorer browser.

Enterprise Features

In the Visual Studio 6.0 Enterprise edition, there are series of enterprise-development tools. These tools make it simpler to build Windows DNA applications. These tools are not development oriented, but support other aspects of the development process, such as versioning and performance.

VISUAL SOURCE SAFE

This is the version-control system for all of your development tasks. It is integrated into all of the development tools of the Microsoft platform. This tool makes it possible to work with your source code, documentation, and other development tools in a team environment.

APPLICATION PERFORMANCE EXPLORER

When developing an application you often need to know the ROI (return on investment). For example, is it better to get a server with 4 processors and 1 gigabyte of memory, or is it more cost effective to buy two 2-processor systems with 512MB of RAM? There is no easy answer, because it depends on the network connections, hard disk types, bus speeds, and processor speeds among other things. The number of parameters on the hardware level is enormous. And then it is possible to tune the services of the operating system to take advantage of the hardware. This adds even more parameters. Calculating the ROI becomes next to impossible.

One way of making the ROI calculation simpler is to use Application Performance Explorer. This is a tool that provides a series of distributed-application scenarios that can be used to test the efficiency of the hardware. These scenarios can then be moved from one machine to another and run again. Based on the results of these scenarios, mathematical calculations can be performed to calculate which hardware gives the best ROI.

VISUAL STUDIO ANALYZER

When testing distributed applications, it is necessary to understand where the bottlenecks are. This is not like using a profiler, because a profiler is specific to a language and requires source code access. Instead, unintrusive testing of the run-time system is accomplished using Visual Studio Analyzer. It can establish a series of counters that can be used to find the bottlenecks in the entire system.

VISUAL MODELER

Every architecture requires a way of designing the entire system. Visual Modeler is a tool that makes it possible to design the system using UML (Unified Modeling Language). UML is a notation used to define classes, methods, and parameters. UML is not specific to a language, and is purely intended for design purposes. Visual Modeler has the capability to generate the model from the source code, and the model can then be used in the implementation phase of the product cycle.

COMPONENT MANAGER

When developing many systems with components, you want to reuse components. These components could be source code, design concepts, or binary components. Component Manager helps manage this process. This tool makes it possible to publish a piece of information that can be reused by someone else.

Component Manager uses Microsoft's repository framework. With Microsoft Repository it is possible to query for specific components based on functionality and feature sets. The repository can be considered a database, but it is more than that, because it is geared toward the development cycle.

Putting It in Perspective

Windows DNA is a way of building applications that support various capabilities, such as transactions, messaging, and data services. Windows DNA and Windows 2000 are joined at the hip. Windows DNA is based on the services exposed by Windows 2000. From the Microsoft point of view, Windows DNA is a marketing tool that presents a way of building applications. However, I find that there are multiple messages and ways of building applications in Windows DNA.

To me, Windows DNA is a way of promoting good programming practices, and the first good programming practice is to define the context of the problem. We took a short look at the context of Windows, which gave us an understanding of why things are as they are on the Windows platform. Doing the same for your own problems explains their history and gives you an understanding of why things have happened the way they have. Based on that understanding, you can develop a sense of direction and better figure out how best to get to your goal.