.NET White Paper Series

.NET:
THE BIG PICTURE

Don Benage

www.gasullivan.com

G. A. Sullivan

is a global e-business solution company. Since 1982, our professionals have consistently delivered complex enterprise solutions and have provided strategic consulting in specific vertical industries. We focus on driving maximum business results from technology investments. As a global leader in the business value and technology of Microsoft .NET, G. A. Sullivan offers agile business solutions that utilize the .NET platform. G. A. Sullivan was among the first in the world to become a Microsoft Gold Certified Partner for E-Commerce Solutions.

G. A. Sullivan delivers scalable, manageable, and reliable e-business solutions for middle market and Fortune 1000 companies. With over 300 professionals across six U. S. and two European locations, G. A. Sullivan offers complete strategic, creative, and technical expertise for challenging e-business initiatives.

G. A. Sullivan combines access to top technical experts and experienced engagement managers with a deep understanding of industry-specific business requirements to ensure solutions that match Line of Business objectives. Vertical industry specialization includes Financial Services (banking, insurance, and securities), Healthcare and HIPAA Compliance, Manufacturing and Distribution, and Government. Technical specialization includes the Microsoft Windows DNA platform as well as the new Microsoft .NET platform and toolset. G. A. Sullivan offers high-level expertise building E-Commerce, Business Intelligence, Knowledge Management, Security and Mobile solutions.

G. A. Sullivan has built a solid team of industry leaders committed to sharing their technical expertise. Members of the G. A. Sullivan team frequently speak at national and international industry conferences. In addition to writing an ongoing series of white papers, the company publishes technical books, writes articles for trade publications, and organizes numerous community development seminars and informative roundtable discussions across the United States and Europe.

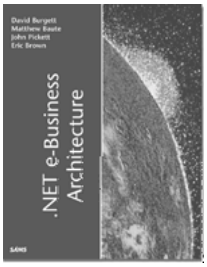
This white paper is intended to assist IT professionals develop their understanding of .NET, and to help them learn how to begin building e-Business and e-Commerce applications using this new technology. Visit G. A. Sullivan's .NET portal site at www.gasTIX.net to get even more details about building enterprise-scale .NET applications. Full source code is made available for download to provide the optimal learning experience.

If you are interested in learning even more about the intricacies of building .NET applications, purchase the SAMS book entitled ".NET e-Business Architecture" (ISBN 0672322196). This book, written by a team of senior designers and developers at G. A. Sullivan, chronicles the best practices and lessons learned in building www.gasTIX.net.

For additional information about G. A. Sullivan visit www.gasullivan.com.

Contents

.NET: The Big Picture



For information, contact:

G. A. Sullivan
55 West Port Plaza
Suite 100
St. Louis, MO 63146-3131
URL: www.gasullivan.com
e-mail: corporate@gasullivan.com
Phone: (314) 213-5600
Fax: (314) 213-5700

First Printing (November, 2001)

.NET: The Big Picture

Introduction

Change is constant in our lives. In every area of human endeavor the ongoing flow of progress continues, and computer technology is no exception. Just as we begin to feel comfortable with a particular platform and toolset, another is introduced and the learning process begins anew. Though we may occasionally lament the effort required to keep up with the latest advancements in technology, the feature-sets of new platforms usually include things we've asked for, or even demanded.

The term ".NET" carries multiple meanings. It is first and foremost an umbrella marketing term used to describe the entire Microsoft platform, much as Windows DNA described the previous platform. But the astute observer will notice that .NET is also used to refer to one or more elements of the overall platform in addition to the entire platform as a whole, which can lead to confusion. The exact meaning of the term depends on the context, and what is most important to the speaker. Here is a short list of different elements that are described by the term .NET:

- The collection of Microsoft's Enterprise Servers
- The Common Language Runtime (CLR)
- The .NET Framework
- An approach to software development emphasizing the role of "Web services" and the use of standard Internet protocols
- The collection of Microsoft's "core" Web services codenamed "Hailstorm", now called .NET My Services

As can be seen in Figure 1, the Microsoft .NET platform encompasses an assortment of technologies, each playing a critical role in a world of interconnected applications.

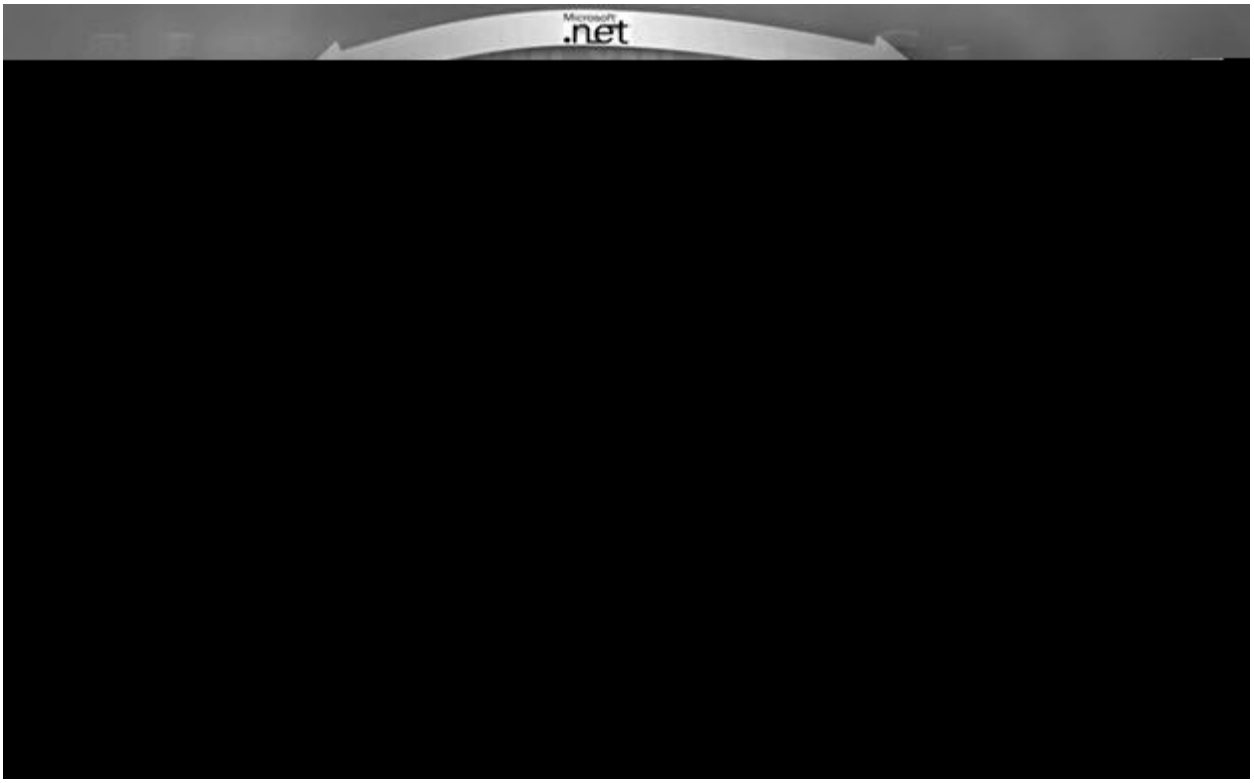


Figure 1. .NET consists of a wide variety of technologies, all driven by the open Internet standard XML.

The Microsoft .NET platform is a comprehensive overhaul in the evolution of Microsoft's software development toolset. Some elements of the platform have been under development for four years, and almost every facet has been changed is some way. This white paper is designed as a thorough introduction to .NET, and how to use it to create e-Business applications and Web sites. A comprehensive sample comprising two Web sites is available at www.gasTIX.net. Welcome to the world of .NET.

The Internet Rules!

As organizations of all sizes and descriptions capture and analyze information, formulate strategies, and ponder their futures, it is clear that the Internet has changed everything. The impact that this multifaceted entity has on our businesses, our schools, our governments, and our lives is profound. This has been stated so often it has already become a cliché, barely worth mentioning.

We are not here to prognosticate on which way the wind is blowing, or the fate of "the dot coms." But to understand the significance and the impetus of .NET one must begin here: The Internet has become a major factor in the way that businesses interact with

suppliers, with customers, and with business partners. It has dramatically altered the availability of information, accelerating the pace of information flow, and increasing customer expectations. The Internet's impact is sometimes obvious, but it has also caused pervasive and subtle changes in the way we think about the world which are only beginning to be fully understood.

A Level Playing Field

What makes the Internet so important? One reason is that the Internet offers a reasonably-level playing field for a large number of participants. The Internet operates with no single organizing force, no single group or individual holding ultimate power. English is not a standard on the worldwide Web, but Hypertext Transfer Protocol (HTTP) and Hypertext Markup Language (HTML) are, as well as Extensible Markup Language (XML). Large corporate entities from leading industrialized nations certainly exert a great deal of influence, but no conglomerate owns the Internet, and it is not governed by the laws of any single country. Of course there are political maneuverings, and people from all over the world engaging in debates and machinations aimed at a favorable outcome for their constituencies and themselves. But by and large the Internet is a place with very low barriers to entry, very little regulation, and lots of opportunity.

Standards: So Many to Choose From

A wise man once said, "The nice thing about standards is there are so many to choose from!" This is true even if we limit our investigation to Web-based protocols, or even those primarily associated with application development and deployment. There are many standards. We like to think of this in a positive way, looking at these standards as many technologies in our toolkit, waiting for the appropriate situation to be used.

Why are standards so important? With so many people participating in this shared endeavor, we must agree on some rudimentary approaches to conducting our conversations, our transactions, and our business. Absolute anarchy serves only the nihilist. If we wish to accomplish something constructive, we need building blocks with which to work. Standard Internet protocols serve as these building blocks.

As a guiding principle, we'd prefer to be "one of the many, not one of the few." In other words, we'd generally like to select mature technologies that are used by many other organizations. And, in general, we'd like standards that are controlled by an independent organization that is not primarily a profit-making entity.

This white paper won't delve deeply into specific protocols, but it will make a few key points about XML and Simple Object Access Protocol (SOAP) because of the central role they play in .NET.

XML

Among its many benefits, XML is a primary enabling technology when we wish to build loosely-coupled systems. To appreciate this benefit, some history is in order.

Among the lessons we've learned over the past ten years is the value of building our systems to be resilient in the face of change. Things always change, usually before we can deploy Version 1.0 of a new system!

As we have struggled to deal with this constant need to adapt, we have learned that if the interface between two systems is highly constrained, any changes must be made at the same time to both systems. This is bad enough, but if we have a single system that connects to many other systems, the result is particularly pernicious. Changing all the related systems in synchronization with one another is nearly impossible. Things become so difficult to change that we may decide instead to institutionalize the current practice. XML provides a flexibility that can solve this problem.

In the past, it was common practice to define a highly structured data transmission protocol. If one application needed to transmit information to another application, the information was often represented in a compressed, binary format. This was done in an effort to reduce the size of the transmission and to maximize the use of relatively scarce bandwidth. An unwelcome offshoot of this approach is a tight coupling between the two applications communicating with one another. If we need to change the content of the transmission, we must change both applications at the same time. For example, software designed to interpret the first six characters of a data transmission as an inventory number is, *by its design*, going to yield a tightly-coupled system.

But suppose we don't build a preconceived notion of the exact positioning of the invoice number into our software. Suppose, instead, that we look for the invoice number between two tags, `<invno>` and `</invno>`, that must appear somewhere within the data stream. Now that's being a bit more reasonable. We might wish to define a schema that represents, for us, the proper syntax for a valid invoice. We might be willing to consider a subset of the information optional. This would allow us to make changes to one system without the other being affected--loose coupling!

We've seen how the design of XML allows us tremendous flexibility and power in the exchange of business documents. The same approach can obviously be used with information of all kinds. This is one of the primary reasons that the use of XML has attracted so much attention. Is there another way to leverage this simple but elegant protocol? Yes, and that way is SOAP.

SOAP

Simple Object Access Protocol (SOAP) is an XML-based protocol that allows applications to exchange structured data over the Web. SOAP ties together disparate services through leveraging industry standard protocols. SOAP allows developers to build XML Web services, which expose programmatic functionality over the Internet.

> For more information about SOAP, visit msdn.microsoft.com/soap.

Let's assume we have a distributed system made up of two or more computers, each running its own operating system, that are connected by various networking protocols (predominantly TCP/IP). For some years there have been passionate debates about object models, application development models, software architectures, and related technologies. As distributed computing and component-based approaches grew in popularity, the competition grew for the hearts and minds of developers. The major vendors involved in providing tools for building enterprise-class applications are often bitter enemies who compete vigorously against one another. The playing field is replete with acronyms representing various approaches: CORBA versus COM/DCOM, Enterprise Java Beans (EJB) versus Windows DNA, and J2EE versus .NET.

Of course, under the surface, there are many similarities in all of these approaches. In particular, all of them offer a way for a process executing on a particular computer to call a remote procedure executing in another process on the same machine, or on an entirely different computer. We need a way to specify the procedure, send one or more arguments, and receive results in the form of return arguments. And every Remote Procedure Call (RPC) should be uniquely resolvable in an unambiguous way, leading us to the correct executable module on the correct computer.

Every platform finds a way to do this; the differences in implementation are perhaps less interesting than the similarities. And yet they *are* different. Systems built with one platform are incompatible with others. This gave rise to such things as COM/CORBA gateways that provide interoperability between two different platforms. But such gateways, living with one foot in each world, suffer from the problems of both, and are nearly always an undesirable compromise. Frequent inconsistencies exist in how either side of the gateway deals with data types and parameters, as well as the inevitable performance hit incurred by the translation process.

Enter the Web, and expectations have changed. The message being sent by the marketplace is this: whatever we build, it must work on the Web, and it must interoperate with other systems (see Figure 2). If difficult and capricious requirements exist for interacting with a given system, the time and expense of doing so will become

too great. All systems must adhere to a reasonably straightforward way of interoperating with external systems, because this is the way organizations wish to conduct business today.

SOAP builds on HTTP security, both HTTPS and X.509 certificates. We can choose which methods to use explicitly. There is no application code included in SOAP messages. Due to its support for HTTP, SOAP is considered firewall-friendly.

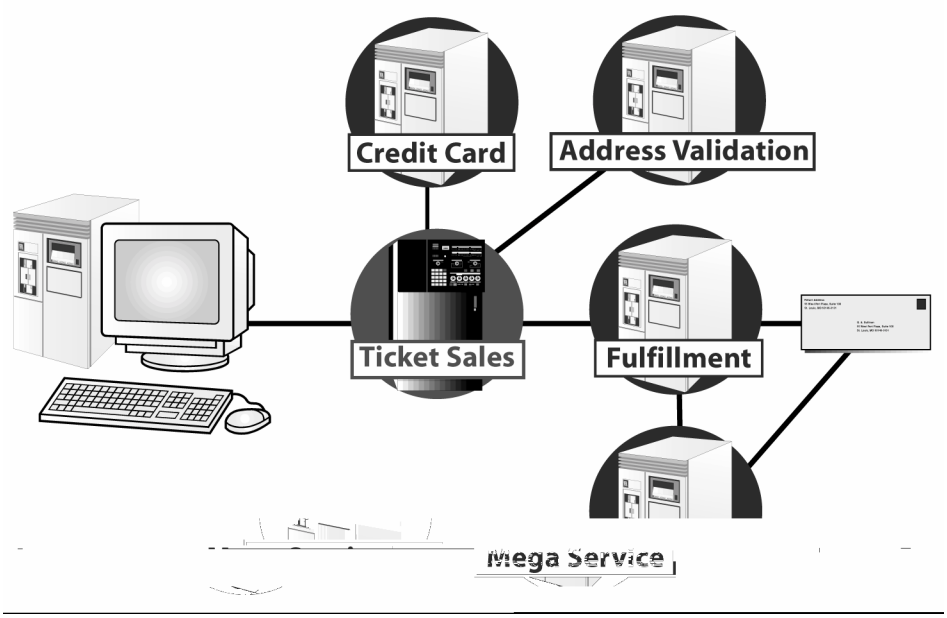


Figure 2. XML Web services enable server-to-server communication across disparate platforms.

Today we see something happening that was unanticipated by most, professionals and pundits alike. An elegant protocol has been devised for calling a remote procedure that runs over standard Internet protocols, is firewall-friendly, and is not "owned" by any single vendor. Take the contents of a remote procedure call, format them with XML, and send this payload over HTTP protocols. What could be simpler? Forget COM/CORBA gateways, façade classes wrapping remote, dissimilar systems and architectures. Let's all speak SOAP!

For more information on the implementation and performance of SOAP calls, see the G. A. Sullivan white paper in this series entitled ".NET Remoting."

Enterprise-Level Attributes

Offering a high degree of interoperability through the use of standard Web-based protocols is clearly a good thing. It is not enough, however. In this age of e-Business and the rise of the "dot coms," sites must be prepared to deliver more. What if we decide to invite the whole world to visit our site following the NFL's Super Bowl or the World Cup soccer finals? What if we want to service a worldwide clientele, in multiple time zones?

The characteristics we need are sometimes described as the "ilities", or, more eloquently, as enterprise class attributes. We've already discussed interoperability. What are the other "ilities"? They include:

- Scalability
- Reliability
- Manageability
- Flexibility

When we say a site should be scalable, we are referring to the ability to handle a large "load"—to perform lots of work. This usually means a large number of concurrent clients, but may also mean relatively fewer clients, each engaged in tasks that require the system to perform much work on their behalf. We need reasonable performance for the first user to visit our site; as the amount of work increases, that performance should not degrade below a well-defined minimum acceptable level.

The site should also be reliable. Depending on the nature of the site we may even wish to approach the elusive "five nines", 99.999% uptime. Since many of the elements we use to build sites do not typically offer this level of reliability, we must usually incorporate redundant elements. By eliminating any single point of failure (an individual element's failure that will bring down the entire site), we can dramatically improve reliability. This goal also impacts our ability to perform scheduled maintenance. No longer can we rely on late night hours after the close of business for operations that will have a major effect on performance, because our business may operate around the clock!

Large organizations are subject to many outside influences. Regulatory and statutory requirements are imposed by governments and competitive pressures are exerted by other organizations. The life cycle of a typical automated system has shrunk steadily over the past few decades. Everyone is in a hurry. In order to respond quickly to change, we must design our systems from the very beginning to support change. We must select an architecture that enables change. Earlier in the paper we discussed loose-coupling as a technique to enable change. Building a system out of discrete components

that encapsulate their functionality is another approach that supports this goal. If we don't address this need, our system is doomed to a very brief useful lifespan.

Perhaps flexibility isn't enough. In many situations, our systems need to go beyond being flexible—they must be *agile.* As our site operates day-by-day, supporting the needs of our organization and navigating the competitive landscape, we may need agility to be successful. Agility combines the idea of flexibility with an element of timeliness, and also connotes the ability to be nimble, to be sure-footed. We can make changes with confidence and proceed successfully when faced with the need to change.

As system architects and developers have pursued these lofty goals, the tools and techniques they use have undergone a constant evolution. The Microsoft .NET platform is seen in this light as the current incarnation of an evolving process, one that continues to this day and beyond. An understanding of how this process has unfolded and its major milestones can be helpful in developing a complete understanding of our current status. Therefore, a brief summary of this process is provided in the next section.

The Evolution of the Programming Model

Developers must learn many things during the course of their career. The first of these is, obviously, at least one programming language. In addition to this, there are broader issues that must be mastered. What are the characteristics of the target platform? What operating system will be running on the computers making up the system? What are its capabilities, and how should they be leveraged to deliver the desired application specific features? Also of interest is the development platform. What editor or environment will be used? What Application Programming Interface(s) (API) or object model(s) are relevant? In other words, what is the programming model?

The .NET Framework has the benefits of Microsoft’s many years of experience building both low-level and rapid application development (RAD) languages and environments. The .NET Framework has been specifically designed to handle the challenges faced by today’s developers.

Clients and Servers

Not so long ago, developers who wrote applications for PCs were targeting individual users sitting at disconnected computers. If they were connected to a network, they might load and save files from a shared server, or print on a shared printer, but very little was required of the developer to utilize these services. Networking system software intercepted calls directed at the local operating system and redirected them to shared devices.

Microsoft Windows and other similar operating environments offered a Graphical User Interface (GUI) that gave developers a whole new programming model-event-driven

programming. The application was essentially a loop that watched for messages sent by the operating system to notify the application of events as they occurred: keyboard activity, mouse movement, mouse clicks, etc.

Over time, networked PCs were increasingly used to deliver applications for groups of users, perhaps a department within an organization. The need for large amounts of structured data grew, and Database Management Systems (DBMS) became popular. This led to the creation of client/server computing, a different way of building applications (see Figure 3). For the first time executable modules which make up the application were executed on more than one computer—the birth of distributed computing. Client/server computing offered important advantages. Server-based databases were capable of enforcing centrally-defined business rules (for example, "Don't allow anyone to delete a customer that still owes us money"), thus allowing more computing power to be applied to a problem.

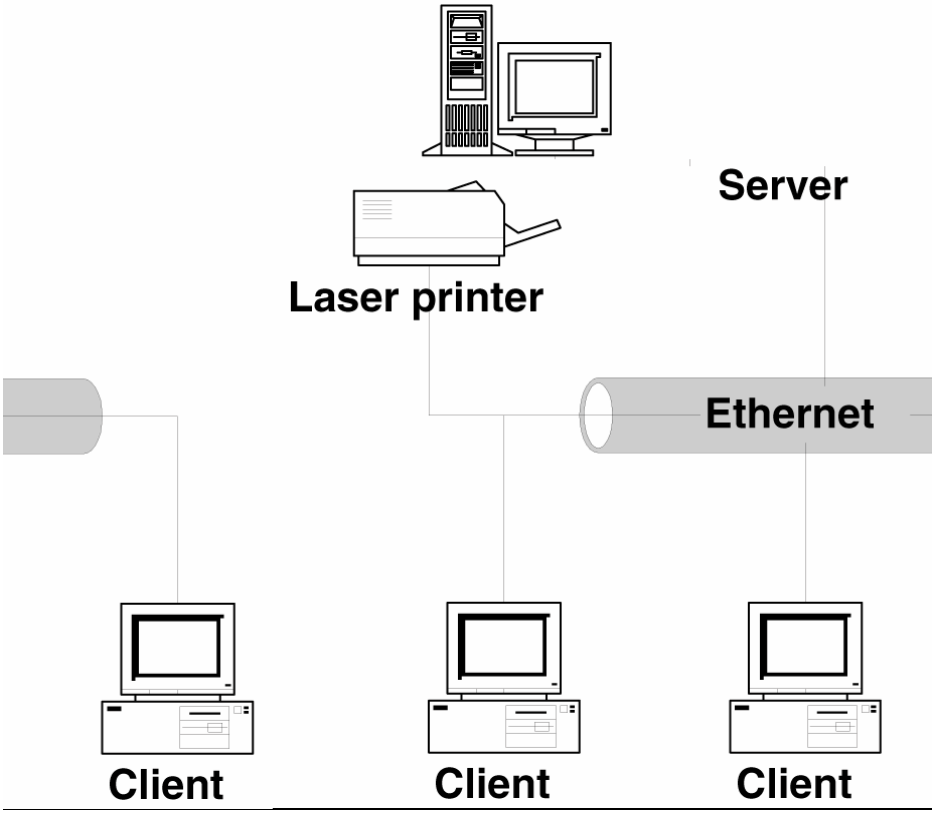


Figure 3. The client/server programming model was a significant departure from monolithic, single-module applications.

As the use of this programming model increased, new problems arose. Among the most serious was the cost of deploying the client portion of the application. In fact, in some cases, the cost of deployment was greater than the cost of creating or acquiring the application. It was very expensive to install a collection of executable modules and

configuration settings on dozens or hundreds of computers. In addition, the protocols that were used for connecting client and server were not designed to work well in a highly-distributed environment like the Internet. The growing use of firewalls presented significant challenges.

For these and other reasons, application developers began exploring the use of Web browsers such as Netscape Navigator and Microsoft's Internet Explorer. At first, using a browser as a "container" for the client-side application meant serious compromises in the feature set available on the client. Far fewer bells and whistles were possible for clients hosted in a browser. Compatibility differences among rapidly changing browsers from different vendors were also a challenge.

Perhaps more importantly, the limitations of using Structured Query Language (SQL) to implement business rules were increasingly apparent. SQL is a non-procedural, set theory-based language that is ideal for specifying data-centric operations on a database. However, SQL is not designed to provide the services of a general purpose programming language.

As these developments were taking place, another paradigm shift was unfolding—the growth of Object Oriented Programming (OOP). OOP offered a programming model with important advantages such as encapsulation and inheritance. Many problems facing developers were easier to solve when viewed from this perspective. OOP also offered a greater opportunity to achieve one of the most elusive goals of large-scale software development-reuse.

Applications Become Sites

Combining elements of client/server and OOP, yet another programming model emerged. Component-based development viewed the application as a collection of modules that could be deployed on networked computer systems. But instead of a 2-tier client/server approach, applications were separated into three logical tiers (see Figure 4), usually referred to as presentation services, mid-tier business logic, and data services. Called 3-tier or, in more complex arrangements, N-tier or multi-tier, this programming model is ideally suited for Web-based environments, as well as for standard application developments. Multi-tier is arguably the predominant programming model for non-scientific applications at the time this paper is being written.

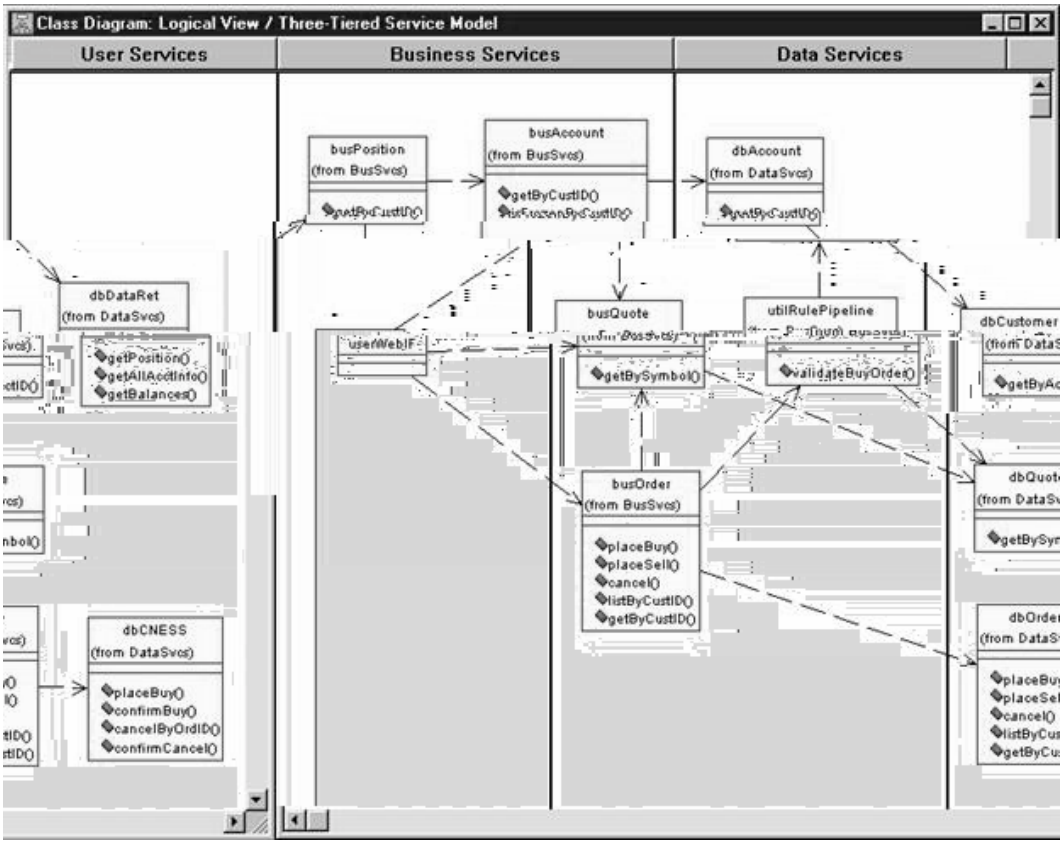


Figure 4. A logical three-tier architecture provided a separation of the business rules embedded in a system.

Presentation services are delivered by the combination of Web browsers and Web servers working in concert with one another. Microsoft's Active Server Pages (ASP) and similar technologies allow application logic on the Web server to execute, ultimately producing HTML for delivery to the browser where it is rendered. Optionally, client-side logic can also be created for execution directly on the client computer. Presentation services are primarily concerned with allowing the user to navigate through the various screens and dialogs that make up the application, accepting input from the user, and presenting results on the client computer's display.

In the middle tier, the rules that define how an organization operates are implemented. For example, we might create components that calculate taxes and apply those taxes only when appropriate. Mid-tier business logic is most often implemented using component-based techniques, which rely on system software to provide services for creating, managing, and destroying components on a server. Microsoft's first offering in this area, the Microsoft Transaction Server (MTS), has evolved into a set of services provided by COM+ for managing components and transactional behavior. This allows the use of object-oriented approaches, and leaves the database to do the things for which it is best suited.

Data services are concerned with the storage and retrieval of information in a persistent store, usually a relational database. Data service components often are designed to encapsulate the exact details of physical data storage and present a "friendly" façade to mid-tier services. This allows the developers of business logic to simply request customer information, for example, and not worry about what tables the information is stored in, the relationships between various tables, and how they are indexed.

It is important to note that there is a distinction between logical tiers and the physical implementation of these elements. It is tempting to think of presentation services occurring on a client computer, mid-tier services executing on a Web server and/or application server, and data services on a database server. In reality, it is a bit more subtle and complicated than this simplified picture, as shown in Figure 5.

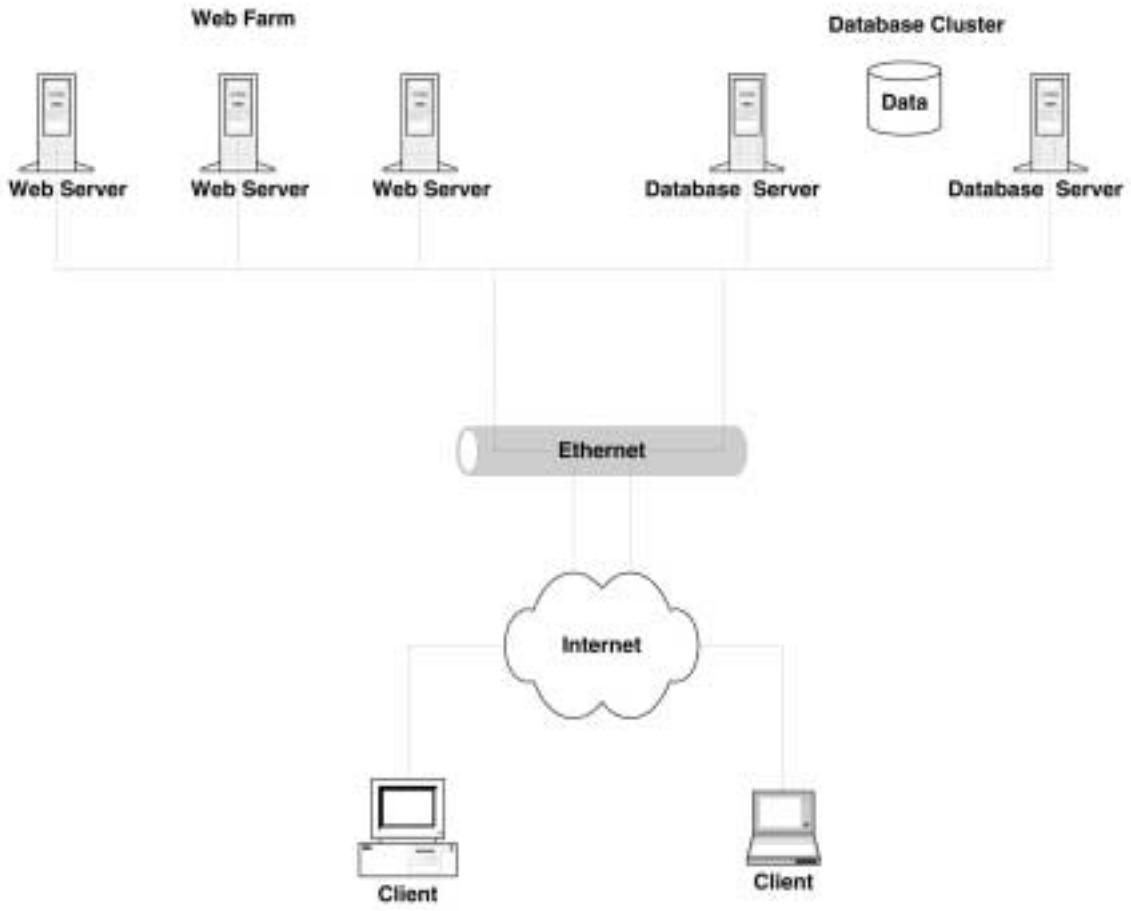


Figure 5. The physical architecture of a large Web site may be made up of many servers working together to function as a single system.

As the use of Web-based approaches has grown, the concept of an "application" is now most often physically implemented as a "site"-a distributed collection of elements that

deliver the functionality of the application. Large Web sites are not implemented on a single computer. The "front door" of the site is often implemented as a Web farm (a collection of load-balanced computers that are all configured in essentially the same way). Middle tier components implementing business rules run either co-resident on the Web servers, or in a physically separate middle tier, which may also be implemented as a load-balanced collection of servers. Finally, the database is most often run on a cluster of two or more machines sharing a high-performance data storage system. So sites that are experienced by users as a single entity are actually run on a distributed set of specially configured servers, each playing a particular role.

Scale Up versus Scale Out

A central and ongoing debate focuses on the issue of how to deliver the maximum computing power to a particular application. Is it better to build very large multi-processor systems with massive amounts of memory, or should collections of relatively normal-sized, commodity-priced servers be used? Both techniques are now possible, and they are referred to as scaling up and scaling out, respectively.

Scaling up is the most common approach to providing maximum power with the database. This is primarily due to the complexity associated with partitioning a large database, made up of many tables, indexes, views, triggers, and other elements, into a collection of more or less uniform servers. It is possible for any reasonably-talented database administrator to do this job manually, but *when* the partitioning scheme needs to be adjusted (and sooner or later it *always will*), we are faced with a difficult and time-consuming job that is almost certain to require scheduled downtime. SQL 2000 offers new support for partitioning a database, and for some applications this makes sense, but as a general rule, it is easier to scale the database up, not out. Scale-up approaches also match well with failover clustering for reliability, particularly important for the transactional nature of most databases.

A scale-out approach is most often used for Web servers, grouping them into a load-balanced Web farm. The elements that make up the Web server's portion of a distributed, component-based application (for example, scripted Web pages, graphics, and configuration files and settings), are deployed in a uniform fashion to each server in the farm. Once this is accomplished, incoming HTTP requests are distributed to one of the servers. Ideally, the amount of work performed by any given server is very close to the amount of work performed by each of the other servers, that is, they share nearly equal loads.

Balancing the Load

A number of approaches to load balancing exist, some hardware-based and some software-based. One of the first approaches was a simple "round-robin" distribution that can be provided by the same Domain Name System (DNS) that directs traffic for

the Internet. Instead of configuring only one entry in the DNS table for a particular domain name, a group of IP addresses could all be associated with a site name (for example, www.gasullivan.com). This approach worked, but didn't offer much resilience when one of the servers went down. DNS would continue to send an equal share of incoming requests to the down (and unavailable) server, causing the client to experience a long wait and eventual timeout of the application or worse.

Networking and router vendors such as Cisco and F5 have offered products for several years that address thee problem of down servers. In addition to distributing incoming requests, these products will typically sense when a server has gone down and remove it (temporarily) from the Web farm. Microsoft acquired and refined similar software-based technology. The Windows Load Balancing Service (WLBS) was first offered as a free download from Microsoft's Web site; then it was subsequently renamed and incorporated into Windows 2000 as Network Load Balancing (NLB). A number of industry pundits have predicted that load-balancing a Web farm is rapidly on its way to becoming a commodity service, and the pricing for this capability continues to drop as additional features are added and performance improves.

Load-balancing is not necessarily a trivial task, however. For example, certain applications may perform a great deal of work using mid-tier components. Such an application might warrant a multi-tier approach that implements a physically distinct collection of mid-tier servers (in other words, the mid-tier components are *not* run on the Web servers). Now we are faced with a more difficult task than equally distributing incoming requests across the server farm. Ideally, we would equally distribute the actual *work*, not just the component requests.

The algorithms for providing this capability are not trivial. Among the problems that must be faced is the need to measure and characterize the load being sustained by a given server. What exactly should be measured? Clearly we want to measure the use of one or more of a server's resources, such as the Central Processing Unit (CPU), memory, or disk access, but which, and in what combination? Also, it is clearly important not to just look at resource utilization at some particular instant, but to measure load over time. But we need to be careful or we will waste precious resources in order to accomplish the measuring.

Finally, there is the question of how to distribute the load. Assuming we know at any given instant which server should receive the next request, what is the actual mechanism that accomplishes this task? Do we send a multi-cast request and depend on the least-busy server to answer first? Or do we provide a traffic manager (potentially a single point of failure) to dole out work? Microsoft has grappled with these issues and offers a Component Load Balancing (CLB) capability in addition to NLB.

A Matter of State

The multi-tier programming model helped solve many problems faced by developers and system architects, but also brought some new challenges. The use of a Web farm to handle very large numbers of users is a great benefit, but as these systems were deployed, tested, and tuned, it became obvious that state management was an issue. Most Web server platforms offer an object model with some type of session construct for managing interaction with a particular user. It is possible to use session variables to store information about a particular session (choices made by the user, user preferences, and other information collectively referred to as "the current state.") Doing so, however, can put a considerable burden on the resources of the server.

An even more difficult problem arises when we allow each request for a new page to be load-balanced. If we force a particular client session to "stick" to a particular server, creating what is called server affinity, we must do extra work to provide this session capability over HTTP, which is essentially a session-less protocol. Furthermore, this will almost certainly cause a less-even distribution of the load.

It would be easier in some respects to allow each request for a new page to be serviced by any server in the Web farm. This would likely provide a more uniform distribution of load and simplify our session management chores. But what about state? If there is any current state information that should be maintained, how is it to be managed when the user is redirected to an entirely different server? As we will explore in some detail in other white papers, the .NET platform offers several alternatives to address this dilemma:

- Use session variables as in the past. Though this limits scalability, it is the fastest alternative with a single Web server.

- Configure a "state server" running as a Windows NT service. A single state server can provide state management for an entire Web farm, providing a balance between performance and scalability. This potentially introduces a single point of failure.

- Configure the Web farm to store state information in SQL Server. Though this adds the most overhead, and compromises performance somewhat, it is the most reliable. The admittedly slower performance will continue to be a problem as the number of users grows very large.

In addition to maintaining state information for a user session, it is also clearly important to be able to identify a particular session. This can be done in a variety of ways, each with its own strengths and weaknesses. The most common method is to use HTTP "cookies" (strings of specially coded information that is stored directly on the user's computer and passed to the server with each request). Cookies are seen as

invasive; they are a privacy concern for some users. Besides, most browsers can be configured to disallow the use of cookies. Therefore, an alternate method must be provided.

The most widely-used alternative to cookies is to imbed the same information directly into the Uniform Resource Locator (URL) that is passed from client to server. Although this circumvents the problem with cookies, it is not as reliable since all session information is lost if a server goes down.

Incorporating Services

The metamorphosis from applications to sites becomes even more interesting when we introduce the concept of XML Web services, as described in the previous section, “SOAP.” XML Web services blur the definition of the term "site." We will explore the concept of software as a service in detail in the next section, but for now we will introduce the idea with a simple example. To avoid confusion, it is perhaps best to note immediately that this use of the word "service" is similar to, but distinct from, its normal English definition: the act of helping or doing work for another or for a community. It is also distinct from a Windows NT-style service, which refers to a specific type of process running on a Windows NT or Windows 2000 computer (similar to a UNIX daemon).

Suppose we are creating a Web site whose purpose is to sell an inventory of items that vary in size and weight. We wish to charge our customers for the items they order, plus offer a range of shipping alternatives, including overnight delivery, as well as normal delivery, that may take several days. How do we calculate the charge for shipping?

We could, of course, review the process a person would use to manually calculate shipping charges for using a particular carrier. We could create a database table representing the current rates for various sizes, weights, and distances. Then we could create one or more business rule components that calculate shipping charges according to the current rules for that carrier. We could then repeat this process for each carrier we wished to use.

The information in these tables, and the rules expressed in the components, will need to be changed periodically, perhaps frequently. A multi-tier approach facilitates this process, but we don't really wish to become quite so involved with the calculation of shipping charges. Perhaps the main concern is that this effort does not represent our core competency. Getting this right is not our primary concern. We want to sell our inventory; we are not in the shipping business.

It would be very nice if we could somehow outsource this calculation to those for whom it is a primary concern, namely the carriers themselves. What if we could send an electronic request for the calculation of a shipping charge to send an item of specified

size and weight from a particular location to another location, and quickly receive a response containing the cost? This is the central idea behind a Web service.

XML Web Services 101

Our shipping calculation example leaves many important details of Web services undefined. Which shipping companies offer such a Web-based service, if any? Exactly what do I send in the request? Do I specify the “to” and “from” locations using a street address, Global Positioning System (GPS) coordinates, or latitude and longitude? In what form will the results be returned? Can I ask for optional information, or must I make multiple requests? And what protocol do I use to send the request?

We could use a variety of protocols to implement XML Web services, but a standard way of doing the job would make sense in this cooperative endeavor. SOAP seems like a natural fit for much of the job we need to accomplish. Our request for a shipping calculation is essentially a remote procedure call. With the addition of a few additional elements, we have the foundation for a powerful new way to build Web-based applications.

For more information on alternatives to SOAP when calling Web services, see the

toolkit also contains a "listener" that will receive incoming SOAP calls and direct them to the appropriate service.

The SOAP Toolkit enables us to:

- Expose a service. If the service we want to expose is a COM component, we can just run the toolkit's "dehydrator," to extract an XML service description.
- Consume a service. Once the service description is generated, the Toolkit's automatic Remote Object Proxy Engine (ROPE) instantiates it as a VB proxy that the programmer can treat as a local COM object.
- Run against a service. When we then run the client application, the proxy uses SOAP to make calls across the network to the Toolkit's Listener application (either an ASP or ISAPI version). The Listener then handles startup of the XML Web service and interaction with the application.

Remember, vendors provide SOAP stacks for other platforms, making SOAP a “broad reach” protocol for communicating with other programming languages, object models or operating systems.

DISCO and UDDI

The Discovery of Web services (DISCO) and Universal Description, Discovery, and Integration (UDDI) are both concerned with finding Web services and interrogating them for usage information.

Before we can access a Web service, we need to know where it is, what its capabilities are, and how this is formatted. Web service discovery is the process of locating and interrogating Web service descriptions.

We can use a program to carry out the discovery when a service publishes a .disco file, an XML document that contains links to other resources that describe the Web service. The following shows an example of a discovery document:

```
<?xml version="1.0" ?>
<disco:discovery  xmlns:disco="http://schemas.xmlsoap.org/disco
xmlns:scl="http://schemas.xmlsoap.org/disco/scl">
  <scl:contractRef ref="http://MyWebServer/UserName.asmx?SDL"/>
</disco:discovery>
```

The discovery document is a container for elements that typically contain links (URLs) to resources that provide discovery information for a Web service. If the URLs are relative, they are assumed to be relative to the location of the discovery document. However, a Web site that implements a Web service need not support discovery. Either another site could be responsible for describing the service, or the service has been created for private use, and there is not a public means of finding the service.

As electronic businesses grow and expand, it becomes more important to have standards that facilitate connectivity and interoperability. Standard descriptions of the business services that a Web site offers and how they can be accessed are needed to implement an effective Web services infrastructure.

A service registry architecture is needed that presents a standard way for businesses to build a registry, query other businesses, and enable those registered businesses to interoperate and share information globally in a distributed manner, just as the Internet was intended to be used. A Web services framework and public registry will enable buyers and sellers and marketplaces, to share information, to connect Web services at low cost, and to support multiple standards.

UDDI was created through the collaborative efforts of Microsoft, Ariba, and IBM. It encompasses a process for the definition and creation of the UDDI Business Registry (UBR) and related artifacts, the operation of the UBR itself, and the eventual licensing of the intellectual property and control of the UBR and future directions to an independent third party.

UDDI is a comprehensive initiative that creates a global, platform-independent, open framework to enable businesses to:

1. Discover each other
2. Define how they interact over the Internet
3. Share information in a global registry that will more rapidly accelerate the global adoption of B2B e-Business

The .NET Passport Service and .NET My Services

A fundamental problem that all applications must address, that of authentication, has not been adequately addressed by the industry. As evidence, consider the number of separate IDs and corresponding passwords an active Internet user must have today. At the time of this writing, a tiny handful of sites allow the use of a more global, shared set of credentials.

The vast majority of sites manage their own unique directory of users, with a wide array of valid password syntax. This is bad for almost everyone. We must either use the same, or similar, passwords on every site (a known evil), or we must write them down and presumably store them in encrypted form, requiring a care in management most people don’t want to provide.

We need one or more global repositories of trust, operated according to some business model, and capable of providing reliable, available, convenient authentication and identity services. Microsoft has launched a collection of Web services, known as .NET My Services, with the .NET Passport service, which has already been operational for some time. Its implementation, however, has been updated to better support .NET development. Additional services will quickly be added to .NET My Services.

.NET Passport consists of three main services:

- Passport Single Sign-in service. Site developers can map sign-in names to information in their databases, which will allow them to offer Passport members a personal Web experience through targeted ads, promotions, and content. Using Passport in this way can help increase customer loyalty, sales, and advertising revenues.

- Kids Passport Service. This service is an optional part of Sign-in service and helps parents protect their children's online privacy by allowing them to determine whether their child can use participating Web sites' services, especially those that collect personally-identifiable information. Kids Passport is the first turnkey solution available to Web sites for managing parental consent and helping sites comply with the Children's Online Privacy Protection Act (COPPA).

- Passport Express Purchase Service. This allows Passport members to store credit card and shipping information in one secured electronic wallet. The purchase process is streamlined because members do not have to retype this information. To make purchases, members click an express purchase button or text link next to the products they want to purchase. Then they simply click to select the credit card and shipping address to use for the purchase. Next the information is sent to the supplier's site to complete the transaction.

The vision guiding .NET My Services is a shared collection of secure, available, professional managed services available to a wide range of computing devices. These “mega-services” complements other custom services we can build using the .NET Framework and the family of .NET Enterprise Servers.

XML Web Services Specifications

In October of 2001, Microsoft released a document outlining extensions to the baseline set of Web service specifications. Baseline XML Web Service Specifications include:

- SOAP
- WSDL
- UDDI

In addition to basic specifications, Microsoft is exploring additional specifications with advanced capabilities. A collection of Global XML Web Service Specifications have now been published. These specifications include:

- WS-Routing: a stateless, SOAP-based protocol for describing an actual message path. It supports one-way and two-way messaging, peer-to-peer, and long running conversations.
- WS-Referral: used in conjunction with WS-Routing, this protocol allows dynamic configuration of the SOAP nodes in a message path, allowing them to delegate processing responsibility to other SOAP nodes.
- WS-Security: providing three primary elements, this protocol offers credential exchange, message integrity, and message confidentiality. It uses a variety of credential mechanisms (e.g. Kerberos tickets or X.509 digital certificates) as the basis for a wide range of security services.
- WS-License: describes the use of various license types to build an electronic "credentials tag" that can be attached to WS-Security messages.

Microsoft, in conjunction with other members of the Internet community, continues the process of gradually adding additional features required by developers.

Cast of Characters

The .NET platform consists of a comprehensive cast of characters, ranging from new ones, like the CLR, to old veterans like Windows and Microsoft Message Queuing (MSMQ). It is important to remember, however, that the .NET platform is not limited to the technologies and products listed here. It will continue to be expanded in the future, adopting new technologies, servers, and programming languages.

CLR: The Common Language Runtime

For software developers, the beating heart at the center of the .NET platform is the CLR. In August of 2001, DevelopMentor hosted their first annual Conference.NET, with over 1000 people in attendance. There, seminal events in the development history of the .NET platform were described for the first time to a large, public audience.

The software that would eventually be called the CLR was first introduced in June of '97 at the Component Object Runtime System Design Review (SDR). Team composition and project startup occurred in November of '97, as work on the SOAP specification was just beginning. The concept of Web services had not yet been conceived.

By November of 2000, the key design goals had been refined to a list that includes primary features we see today in the CLR:

- New memory management, featuring the use of garbage collection rather than reference counting and deterministic finalization
- Just in Time (JIT) Compilation. Various designs were implemented, explored, and refined, including Optimizing JIT and Fast JIT models
- Thread and Process Management
- Virtual Object System
- Remoting
- Common Type System (CTS)
- Code Access Security (CAS)
- Support for debugging and profiling
- Unmanaged code support
- Simplified, flexible deployment model

The CLR enables a development environment with many desirable features. These include cross-language integration, a strongly-typed shared type system, self-describing components, simplified deployment and versioning, and integrated security services.

Since the CLR is used to load code, create objects, and make method calls, it can perform security checks and enforce policy as managed code is loaded and executed. Code access security allows the developer to specify the permissions that a piece of code needs in order to execute. For example, permission may be needed to read a file or

access environment settings. This information is stored at the assembly level, along with information about the identity of the code.

In addition to code access security, the runtime supports role-based security. This builds on the same permissions model as code access security, except permissions are now based on user identity rather than code identity. Roles represent categories of users and can be defined at development time and assigned at deployment time. Policies are defined for each role. At runtime, the identity of the user on whose behalf the code is running is determined. The runtime checks what role(s) are associated with the user, and then grants permissions based on those roles.

The CLR encourages moving away from scripted languages compiled at run time to the concept of Managed code. Managed code means a clearly defined level of cooperation between an executing component and the CLR itself. Responsibility for many tasks, like creating objects and making method calls, is delegated to the CLR, which provides additional services to the executing component.

Source code is compiled into Microsoft Intermediate Language (MSIL) that is then consumed by the CLR. In addition to MSIL, .NET compilers also produce metadata. This is information the CLR uses to carry out a variety of actions such as locate and load class types in a file, lay out object instances in memory, and resolve method invocations and field references.

The metadata that the .Net Framework compliers produce includes class definitions and configuration information. It allows an application to totally describe itself. The metadata includes information about dependencies that can be used as part of versioning policies to indicate which version of a particular component should be used when running a particular component. This greatly eases the complexities of application installation and DLL versioning.

The .NET Framework introduces the assembly, a group of resources and types, along with metadata about those resources and types, which is deployed as a unit. The metadata is called an assembly manifest and includes information such as a list of types and resources visible outside the assembly. By using manifests, the developer has support for per-application policies to control the locating and loading of components.

Through the use of MSIL and assembly metadata, code written in many languages can be debugged together, with no loss of information or control as we step through multi-language projects. An exception thrown by a component written in one language can be dealt with using a component written in another language. Cross-language inheritance is supported.

Assemblies can be made private to an application or can be shared by multiple applications. Multiple versions of an assembly can be deployed on a machine at the

same time. Application configuration information defines where to look for assemblies, thus the runtime can load different versions of the same assembly for two different applications that are running concurrently. This means that installing an application now becomes as simple as copying the necessary files to a directory tree, provided security permissions are satisfied.

One of the most important benefits of the CLR is that it provides high-level services to developers that are not restricted to a particular hardware platform or Operating System. Code written to the CLR does not need to be re-written to run on different platforms and under different Operating Systems. One immediate benefit is that the same code can run on all of the various flavors of 32-bit Windows and also 64-bit Windows. It extends the familiar ideas of runtime libraries, template libraries, and the Java Virtual Machine. It means that developers can reuse more code and have to develop less in the way of the more basic services themselves. The CLR and .NET simplify the programming model and make it more consistent.

Initially, the CLR will support four languages: Visual Basic .NET, Visual C++ .NET, Visual C# .NET, and Visual J# .NET. There are also many companies working on other compilers that produce managed code for languages including COBOL, Pascal and Perl.

ASP.NET and Internet Information Server (IIS)

The success of Windows servers has been fueled primarily by their use as application servers and Web servers. Although Microsoft and its competitors argue over market share numbers, Internet Information Server (IIS) is certainly one of the most widely-used Web server platforms on the Internet. Used in conjunction with the family of .NET Enterprise Servers, IIS provides a feature-rich environment for running Internet and intranet applications.

Web servers play a central role in current application development practice. Most often, a client connects to a Web server using a Web browser. Typically the Web server will be running an ASP.NET application hosted on IIS. The application may then connect to, and consume the services of, various .NET Enterprise Servers and megaservices such as .NET My Services, returning the results to the client. The results will often be returned in some form of an XML-based document.

The .NET Enterprise Server family is described in the next section of this white paper. The .NET Servers are listed here in alphabetical order to serve as a convenient reference:

- Application Center 2000
- BizTalk Server 2000
- Commerce Server 2000
- Content Management Server 2001
- Exchange Server 2000
- Host Integration Server 2000
- Internet Security and Acceleration Server 2000
- Mobile Information Server 2001
- SharePoint Portal Server 2001
- SQL Server 2000

ASP.NET is an updated version of Active Server Pages (ASP) which is a Web-based application environment that has been available for some time on Windows NT and Windows 2000 servers. This new version allows applications to be written in all CLR-based languages, including Visual Basic .NET (VB.NET) and Visual C# .NET (C#). Previous versions of ASP were limited to the use of scripting languages with limited feature sets.

ASP.NET also boasts dramatically improved performance due to support for enhanced caching capabilities, and great improvements in the separation of code and design content, code deployment, configuration, and state management. ASP.NET has been designed to allow deployment on any Web server platform, but certainly IIS will be used the vast majority of the time. No plans to release a version for another platform have been announced.

IIS allows an organization to host multiple Web sites on a single computer. Process throttling lets administrators limit the amount of CPU time a Web application or site can use during a time interval to ensure that processor time is available to other Web sites or to non-Web applications. Per Web site bandwidth, throttling allows administrators to regulate the amount of server bandwidth each site uses. This enables an ISP, for example, to guarantee a predetermined amount of bandwidth to each site.

Web-based authentication can be fully integrated with a Kerberos security infrastructure. The Kerberos Version 5 authentication protocol, which provides fast, single logon, can replace NT LAN Manager (NTLM) as the primary security protocol for access to resources within or across Windows 2000 domains. In addition, Windows servers also support the following standard authentication protocols, which are applicable to Web-based users and ordinary network users alike:

- Digest Authentication: An authentication standard of the World Wide Web Consortium (W3C).

- Server-Gated Cryptography (SGC): A system used by financial institutions to transmit private documents via the Internet.

- Fortezza: A U.S. government security standard.

Digest Authentication enables secure authentication of users across proxy servers and firewalls. It offers the same features as basic authentication, but improves on it by "hashing" the password traveling over the Internet, instead of transmitting it as clear text. For those who choose not to use Digest Authentication, Anonymous, HTTP Basic, and integrated Windows authentication (formerly called Windows NT Challenge/Response authentication), and NTLM authentication are still available.

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) provide a secure way to exchange information between clients and Web servers. In addition, SSL and TLS provide a way for the server to verify client identity before logon. Beginning with IIS 5.0, programmers can track users through their sites. Also, IIS 5.0 lets administrators control access to system resources based on X.509 client certificates. It is anticipated that a new release, IIS 6.0, will be included in the new .NET server products planned for release sometime in 2002.

.NET Enterprise Services (COM+)

The collection of services that were called COM+ services are now referred to collectively as .NET Enterprise Services. Originally developed by the COM team, these run-time services are now “baked-in”, tightly integrated directly into all server versions of Windows operating systems, with some optional features and capabilities only available on the high-end editions. These include support for transactions, object context and call context management, object pooling, and data connection management.

Beginning with Microsoft Transaction Server (MTS), Microsoft has steadily refined its mid-tier component strategy, enhancing object pooling performance and control. Transactional behavior can still be indicated by the developer at design time (through the same mechanisms introduced with Windows DNA), and now with the use of attributes in CLR-based languages. Settings can be overridden by the application

administrator in the deployed environment using the appropriate Microsoft Management Console (MMC) based administrative console.

The .NET Framework

Using Microsoft's previous development tools, Visual C++ developers programmed against one set of Windows APIs, while Visual Basic developers programmed against another. The ODBC API was an interface programmed against for data access, and the CDO library was the API used for e-mail. Dozens of other APIs were released by separate Microsoft product groups at different times, each providing an interface into bits of functionality provided by the underlying Windows operating system. Releasing a multitude of APIs in this fashion meant that there was no clear relation between the many different interfaces, and there was no easy way for an application programmer to locate specific functionality that was needed. Additionally, when a program that used a specific API was deployed, programmers needed to make sure that a specific DLL was deployed and registered on the target machine.

The .NET Framework base class library is designed to remedy these issues. All functionality is now contained within the same, universal API set and is made available to developers of any .NET programming language. No longer do C++ programmers use one set of APIs while VB programmers use another. Each and every development language uses the same interfaces, making switching between languages a matter of syntax rather than functionality. Microsoft now categorizes functionality into *namespaces*, making it easier to find the functionality needed for an application.

At the root level of the .NET Framework class library is the `System` namespace. Access to Web Forms, Windows Forms, data access, networking, messaging, and all other functionality is contained in some namespace underneath `System`. For example, `System.Web.UI` contains functionality to create Web controls on an ASP.NET page. `System.Data` contains functionality that lets us connect to databases and manipulate data using ADO.NET. Although the different namespaces are physically implemented in multiple assemblies (.DLLs), from a programmer's point of view, they are now more integrated and easier to navigate.

Because the target machine on which a .NET application will be deployed must have the .NET Framework installed, we no longer have to worry about making sure a specific DLL is delivered and registered. Thus, the design of the .NET Framework base class library thus makes life easier for both administrators and developers.

Microsoft Message Queuing

Microsoft Message Queuing (MSMQ) is a technology introduced in Windows NT 4.0. The use of queuing as an element in application design is not new. A number of scenarios exist where a queue can play an important role in making systems more

reliable and more available. For example, perhaps an application must communicate with another computer system that is frequently offline or unavailable. By introducing a queue, the application can simply store requests or messages destined for the other system until that system is once again available. More generally, a queue may be useful in any scenario involving asynchronous or delayed processing.

MSMQ features include:

- COM-based access. MSMQ services are readily accessible through COM interfaces, making them usable for both Windows DNA and .NET applications now. It is expected that tighter integration with the .NET platform will be provided in the next release.

- Support for Transactions. Both transactional and non-transactional queues may be created. MSMQ operations can be included in transactions with full ACID properties to preserve data integrity and provide simplified error recovery semantics.

- Prioritization. All messages may not be equally important, so MSMQ allows priorities to be assigned to messages and queues. It then manages routing and delivery based on these priorities.

- Notification. MSMQ can notify a sending application that messages were (or were not) received and processed correctly. Rather than polling to find out if a message was properly handled, this capability lets applications know if failures occurred, allowing them to take corrective action as needed.

- Built-in data integrity, data privacy, and digital signature services. Security continues to grow in importance, and MSMQ provides strong security support. MSMQ can digitally sign and encrypt messages for transfer across the network, protecting messages from being viewed or changed during transmission, even when sent over public networks such as the Internet.

- Simplified application integration. MSMQ can simplify the design of complex systems by providing an easy-to-use messaging infrastructure that can connect disparate systems with different operational semantics.

- Network protocol independence. MSMQ features do not depend on the use of a particular network protocol (e.g. TCP/IP). Applications running on systems using different protocols can share the same queues provided the server or servers running MSMQ are configured with multiple (and appropriate) protocols.

- Message journaling. An optional copy of messages sent or received by applications can be kept, providing an audit trail. Journaling can also simplify recovery from certain types of failure.

Flexible, reliable asynchronous communications between component-based applications are critical for building large applications, especially when those applications must communicate with disparate systems running other platforms. MSMQ's services are an important compliment to .NET Enterprise Services and will remain a key design element as the platform continues to evolve.

ADO.NET

It should come as no surprise that a main function on any e-Business application is to read data from and write data to a database, whether it be SQL Server 2000 or Oracle. Databases capture orders, maintain customer lists, aggregate data for OLAP reporting, and archive historical information. It goes without saying, then, that .NET e-Business developers need a strong programming interface for manipulating and processing such data. This interface is ADO.NET, a significant evolution of Microsoft's ActiveX Data Objects (ADO) technology that has been around for some time.

ADO.NET is based on XML, as is the .NET platform as a whole. The concept of loose coupling and data interchange is key to the .NET concept, and therefore the data access layer should reflect this concept. ADO.NET does in fact reflect this design goal, and the XML support built into the technology is evident. Not only is XML support pervasive, but a whole new way of manipulating data in memory is supported. ADO.NET allows developers to represent multiple tables and their relationships in memory for processing on the client side using a `DataSet`. This is an important evolution of the technology, providing a great deal of power to the developer that was previously unavailable.

A key design goal of many .NET e-Business applications is high scalability. ADO.NET supports this with disconnected recordsets. This prevents a middle-tier object from having to keep a database connection open, which can be a significant hindrance to scalability. The `System.Data.SqlClient` namespace is also provided for as an optimized interface when working with SQL Server.

Windows 2000 and the .NET Servers

The power of a distributed application consists of two major factors: the ability of distributed components to communicate, and the power of those components individually. The recent release of Windows XP marks a new chapter in the evolution of operating systems. Windows XP eases the migration path of both professional and home users to a single operating system. This defines a path into the future where users

can have access distributed components in an identical fashion from both their work and home computers.

In addition to the benefits of a single code base, Windows XP incorporates the Internet and networking into the operating system better than any previous version. This allows more home users to access the Internet and greatly decreases the effort required to setup and administer even complicated networks. Windows XP inherently understands and utilizes XML. Much of the .NET platform depends on XML, from XML Web services to SQL Server XML result sets. Applications run on Windows XP will benefit from using the operating system's built-in XML parser, rather than depending on a plug-in version. This will also greatly ease application deployment in the future. Since Windows XP is currently only available in Home and Professional versions, we must also take a look at the more advanced feature sets of Windows 2000 available today.

Datacenter Server

Even the most ardent admirers of the Windows family of operating systems have occasionally questioned its reliability. Software developers who have spent a significant amount of time working with Windows have probably experienced the "blue screen of death" (a screen displaying the contents of various memory locations and CPU registers that Windows displays when a fatal crash occurs). One of the fundamental problems in trying to address this serious problem has been the large number of hardware devices and corresponding device drivers supported by Windows. Device drivers, by their nature, must directly access hardware and can cause fatal errors if not properly designed and thoroughly tested.

In order to address this issue, Microsoft has teamed up with hardware vendors to provide the Windows 2000 Datacenter Server, which includes device drivers that have passed the very rigorous Datacenter Server Hardware Compatibility Tests. This version of Windows is only available directly from the hardware vendor. This is further augmented by an integrated approach to support that includes both hardware and software, and application certification specifically designed for this platform.

In addition to addressing reliability concerns, Datacenter Server includes features that are important for very large installations and applications. It supports up to 32 processors and up to 64 GB of RAM. In addition to providing all the services and features of Windows 2000 Server and Advanced Server, Datacenter Server also provides:

- Process Control, a new tool in Datacenter Server that uses job objects to organize, control, and manage the processes on a system, along with the resources they use.

- Four-node failover clustering support based on a "Shared Nothing" model.

- Enterprise Memory Architecture (EMA). EMA supports two distinct models for enhancing the use of memory:

    - Physical Address Extension (PAE). PAE allows the operating system kernel to use all available physical memory, including memory above 4 GB (up to 64 GB). Applications need not be rewritten to benefit from the reduced paging provided by this support. .However, individual applications cannot utilize the full address space without using a new API called Address Windowing Extensions (AWE).

    - Application-Memory Tuning (sometimes referred to as 4-gigabyte tuning or 4GT).

- Winsock Direct, a feature that provides substantial performance improvement for Winsock applications without requiring modification. Winsock Direct is essentially an optimized TCP/IP stack capable of very efficient, high-bandwidth, low-latency messaging. This reduces processor utilization, conserving processor time for application use.

Server and Advanced Server

Windows 2000 is based on the earlier Windows NT operating system. All members of the Windows 2000 family share a core feature set, with additional functionality and options added to higher end versions.

At the time of this writing, Windows XP has been released in both Home and Professional editions; however Server and Advanced Server editions are not yet available. It is anticipated that when they are released, they will provide specific enhancements to directly support the .NET platform. The exact naming of these editions has not yet been announced, although industry watchers expect that there will still be Server and Advanced Server editions. Until updated server operating systems are released, the Windows 2000 server family remains the available offering.

Windows 2000 Server is the entry-level version of the family. It supports between one and four processors and a maximum of 4GB of RAM. The base feature set, which is shared by all members of the Server family, includes:

- Enhanced protection of operating system files and registry settings designed to prevent inadvertent destruction, removal, or replacement of key elements of Windows 2000.

- Fewer operations requiring a system reboot. Administrators have requested attention to this area for some time. Whether an outage is due to system failure or a scheduled system upgrade, a reboot still makes the server unavailable and disrupts connected sessions.

- Fast Recovery from System Failure. If the system does fail, Windows 2000 includes an integrated set of features that speed recovery.

Windows 2000 Advanced Server builds on the feature set of the standard version of Windows 2000 Server, and includes additional features to support applications requiring higher levels of scalability and availability. Windows 2000 Server supports up to 8 processors and up to 8 GB of RAM. Among the additional features are:

- Two-node failover clustering support. In this configuration, two systems share a hard disk subsystem. The machines need not be identically configured or be running the same tasks. When one server fails, the other picks up its load.

- Up to 32-node NLB support. Another form of clustering, this capability is most often exploited to build Web farms.

- Rolling Upgrade Support. Using Microsoft Clustering Services (MSCS) and NLB, downtime caused by planned maintenance or upgrades is avoided by using rolling upgrades. Applications are migrated onto one node. The other node is then upgraded and the workload is migrated back without taking the application off line.

The Microsoft .NET Enterprise Servers

ASP.NET and IIS provide the engine for Web-based development efforts, but there are many other services that are required to build enterprise applications. These include relational database management, support for mobile communications access, integration with host systems, and advanced management tools. A growing set of services are provided to .NET applications by a collection of .NET Enterprise Servers. The current collection is described below, but it is anticipated that additional servers (and updated versions of these products)) will be added over time.

Application Center 2000

Application Center 2000 is a tool for deploying and managing Web applications across clusters of servers. It provides software-scaling services that allow applications to achieve very high scalability (both peak load and sustained load as characterized by number of users or resource utilization), and mission-critical levels of availability. It also reduces operational costs and complexity.

Application Center allows us to manage our applications using a single high-level definition. This definition includes all of its content, components, and configuration settings. All configuration changes are made through a standard MMC snap-in. Performance and event log data from one or all participating machines can be viewed from one place, keeping application content and configuration settings consistent and

synchronized across all servers in a cluster. Synchronization can be automated with timed or on demand updates. Rudimentary “self-healing” capabilities can be used to improve reliability.

Application Center can provide change management for applications. We can deploy versions from a development cluster to testing, then to staging, and finally to production, automatically reducing downtime. We can manually rollback a failed deployment, and, when necessary, replicate the "last known good" application image. This facilitates rapid recovery from failed deployments.

Windows 2000 Advanced Server contains NLB services that balance network traffic across multiple servers in a Web farm. Application Center configures and controls NLB. It also configures and controls COM component execution across multiple servers through its Component Load Balancing (CLB) services.

Application Center provides advanced fault tolerance. A properly-designed site can withstand software and hardware failures at any single point in the system without disrupting application service. The performance and health of the system can be monitored from a single console. Performance data for any server in the cluster, or for the entire cluster, can be gathered and analyzed.

BizTalk Server 2000

BizTalk Server 2000 is a server platform designed to make it easier to integrate business processes across company boundaries to include partners, vendors, and customers. This integration is all done using XML-based documents. This document exchange infrastructure allows secure and reliable company relationships to be quickly implemented, independent of operating system, programming model, or programming language.

The foundation of BizTalk Server 2000 is its rules-based document routing, transformation, and tracking capability. SQL Server provides high-performance storage and easy-to-schedule transformation capabilities for data from BizTalk Server.

BizTalk adheres to a set of guidelines defining the structure of business documents, such as purchase orders and invoices. Each business document definition is defined by a schema. The manner in which one company publishes the information on its purchase order forms may be totally different from what a supplier expects. Using XML, the information contained on a form is easily mapped to a particular schema-defined layout, and then sent to the partner company.

BizTalk Server 2000 contains a host of rich graphical tools for building XML schema, performing schema transformation, establishing trading partner relationships over the Internet, and tracking and analyzing data and documents that are exchanged. It also

contains graphical tools that make it easy for business analysts and application developers to model and implement company specific solutions.

BizTalk Server 2000 is built on a foundation of public standards and specifications, including XML, HTTP, and EDI (Electronic Data Interface). It incorporates security standards such as public key encryption and digital signatures. BizTalk Server 2000 can take an EDI data stream, translate the information it contains into XML, and send it to a trading partner. This makes it a useful tool in EDI integration scenarios.

Commerce Server 2000

Commerce Server 2000 is a tool for managing e-commerce Web sites. To support the operation of an effective site, it helps the site operator to attract and engage customers through the use of scheduled online marketing and advertising campaigns and targeted promotions. Commerce Server also assists with management of the transactions generated on the site. Finally, it analyzes the usage patterns and reactions to the content on the site.

Commerce Server has tools that allow the site to be personalized for particular customer types in various ways:

- A secure and scalable environment is provided for order capture and management.

- Marketing actions and promotions can be built to reward frequent shoppers. For example, two-for-the-price-of-one promotions can easily be set up. The end user experience can be modified to greet returning customers personally.

- Customer profiles can be built and maintained and knowledge of our customers' buying habits can be used to improve the site.

- Product catalogs and price lists can be built and used for business partners. Millions of products can be managed.

- Business process pipelines provide a framework for defining and linking together stages of a business process. Analogous to the process of completing a wizard, pipelines allow tailored processing of orders, advertising, merchandizing, content selection, and direct mailing.

- Decision support tools are available to help understand and refine an online business. This incorporates all relevant information, including click-stream data.

As with any business, it is very important to figure out what sells and what doesn't. Which of the various product lines on offer are visitors interested in? Commerce Server actively manages the Web content and analyzes the usage data, providing answers to

this sort of question. To uncover important trends about activity on the site, pre-configured analysis reports can be used, or site managers can build custom reports for mining usage data.

Content Management Server 2001

Content Management Server 2001 is a relative newcomer to the .NET Enterprise Server family. Large Web sites require a constant flow of new content to interest users and to keep up with necessary changes. Many people who are responsible for creating this content are not members of a technical staff in an IT department. Content Management Server simplifies the management, scheduling, and flow of content. It lowers management costs and organizes the process of keeping an application or site refreshed with up-to-date information. Content Management Server is designed to support sites with high scalability and availability requirements.

Exchange Server 2000

Exchange Server 2000 is the latest release of the Exchange messaging platform. Exchange Server is closely integrated with the Windows 2000 operating system and Active Directory. Using the new Web Storage System to add the accessibility and openness of the Web to the reliability and scalability of Exchange Server, Exchange 2000 Conferencing Server provides a platform for complete data, audio, and video conferencing services, thus establishing a foundation for new avenues of collaboration.

The Web Storage System includes built-in indexing for high-speed searches, enabling users to find content quickly and easily. All content in the Web Storage System is indexed, including messages, standalone documents, contacts, tasks, calendar items, and collaboration data. Indexing is accomplished by an indexing "crawl" of the content in the Web Storage System, using the same technology used in Internet Information Services (IIS) and SQL Server 7.0. Users of Outlook 2000 can search for documents in the Web Storage System as easily as they can search for e-mail messages, thus increasing user productivity.

Exchange Server's native support for XML and HTTP is typical of the standards support included in .NET Enterprise Servers. It also includes support for OLE DB 2.5 to integrate with SQL Server, and other OLE DB compliant products.

Exchange 2000 includes two OLE DB providers, a remote provider for server access from client applications, such as Outlook 2000, and a local provider implemented natively in Exchange for high-performance COM access from applications, including virus scanning programs and workflow engines. Application designers can also use ADO to navigate, query, filter, and sort Exchange Server data. This allows developers familiar with developing SQL applications to easily write applications that use data stored in the Web Storage System, using the same tools and expertise.

Host Integration Server 2000

Host Integration Server (HIS) 2000 is the latest release of the product formerly named Microsoft SNA Server. It includes a comprehensive set of integration components for connecting host-based data and transactions with new Web-based applications. Host Integration Server provides traditional gateway integration, data access and database replication, as well as integration of both tightly and loosely-coupled systems. In most scenarios, it utilizes only Windows-based code with no host footprint, a fact appreciated by most host administrators.

HIS provides a wide variety of integration technologies, enabling developers to integrate various host technologies. It provides services that extend a Microsoft interface to traditionally non-Microsoft platforms, including:

- SNA or TCP Protocols: HIS inherits the strengths of predecessor SNA Server for gateway connectivity. While the protocol trend is clearly towards TCP, many host shops still require support for SNA-based applications.

- The OLE DB for DB2 Provider that provides COM+ objects with data access and distributed transactional integration with DB2. These give developers the flexibility to quickly and easily build n-tier applications that integrate COM+ with IBM's DB2, CICS and IMS transactions.

- COMTI (COM Transaction Integrator) allows customers to expose mainframe CICS and IMS transactions as COM objects. COMTI even includes support for distributed two-phase commit. All this is done without requiring any changes to the host application. (This technology has been shipping in Microsoft SNA Server since Version 4.0 in November, 1997.)

- Integrating MSMQ with MQ Series using the MSMQ Bridge, which is responsible for routing application messages between Microsoft's MSMQ and IBM's MQ Series. The bridge allows MSMQ to communicate and exchange messages directly with MQ Series platforms.

HIS 2000 provides these services either via SNA or TCP/IP, without requiring any changes to the host or any host code to be loaded. HIS 2000 supports the native host interfaces and standards and is therefore non-intrusive.

XML integration in HIS is provided via integration with BizTalk, which has the ability to transform and manipulate documents. Once the incoming information is transformed, BizTalk can utilize HIS 2000 for either of the two following integration methods:

- Synchronous or COM+-based Integration: BTS 2000 supports a COM interface (Ipipeline) that can be used to invoke a COM+ business process. This process in turn executes either a DB2 SQL statement via OLE DB or a CICS/IMS transaction via COMTI.

- Message-Oriented Middleware (MOM)-based Integration: HIS 2000 provides a MSMQ to MQSeries Bridge to allow BizTalk to easily and effectively exchange documents asynchronously via MOM.

As an SNA gateway solution, HIS 2000 runs on Windows NT Server and Windows 2000 Server to connect PC-based local area networks (LANs) to IBM System/390 mainframe and AS/400 midrange systems. HIS 2000 enables users of the leading desktop systems (including Windows 2000 Professional, Windows NT Workstation, Windows 95, Windows 3.x, Macintosh, UNIX, MS-DOS, and IBM OS/2)) to share resources on mainframes and AS/400s without installing SNA protocols on the PC, or deploying any software on the host.

The HIS 2000 gateway functions handle the translation between the PC LAN transport protocol—whether TCP/IP, IPX/SPX, NetBEUI, or other supported protocols—and the SNA LU protocols running to the host. By allowing each machine to run its native protocols, SNA Server minimizes resource requirements on each PC and on the host system. The configuration also reduces administrative costs by enabling centralized management of the gateways.

Internet Security and Acceleration Server 2000

Internet Security and Acceleration Server 2000 (ISA Server) is Microsoft's new enterprise firewall and Web-caching server. The main problems that need to be addressed by system administrators and business managers alike include:

-

- How to control bandwidth usage and costs? When thousands of requests go out to the Internet for the same static content it is inefficient and costly.

- How to manage a network as simply as possible? Managing a network and keeping it secure are complex issues. Solving some of the above points can lead to more problems. For example, if we add a cache to improve performance and reduce connection costs, a separate set of resources and expertise are required to manage it.

ISA Server includes the following major capabilities:

- ISA Server securely routes requests and responses between the Internet and client computers on a private network, serving as a firewall, separating a protected private network from the Internet. ISA Server can help defend the network from hackers, unauthorized access, and virus attacks, by filtering and analyzing traffic prior to routing.

- ISA Server provides a high-performance Web Cache Server that improves Web site access performance for network clients by storing frequently requested Internet sites locally. It can be used to speed up performance for internal users accessing the Internet, or external clients accessing the Web server.

- ISA Server combines the benefits of both a firewall and a Web cache with integrated management, applying the same access policies to the firewall and the cache. It takes advantage of Windows server features such as QoS, VPN, advanced authentication, NAT and Active Directory.

For more information about ISA Server, visit www.microsoft.com/security.

Mobile Information Server 2001

Mobile Information Server 2001 is a platform for extending the reach of Microsoft .NET Enterprise applications, enterprise data, and intranet content to the mobile user. This server product will allow mobile users to stay connected to their corporate intranets and applications, using devices such as hand-held PCs or mobile phones. Mobile users will be able to securely access their e-mail, contacts, calendar, tasks, or any intranet line-of-business application. Mobile Information Server integrates with the .NET Enterprise Servers with multiple hardware platforms, and will also integrate with new speech technologies, allowing the voice to control a new user interface.

SharePoint Portal Server 2001

SharePoint Portal Server is a new document management portal solution that allows users to find, share, and publish information easily. It provides features such as document management, search, subscriptions, and online discussions. SharePoint Portal Server becomes the single source for information, combining normal office documents with Web pages and emails. New documents are saved and checked in and out, while document stores capture relevant metadata.

The document management capabilities allow changes in multiple drafts to be tracked as a document is edited, reviewed, and approved. SharePoint Portal Server is designed around industry and Internet standards, such as OLE DB, XML, and Microsoft Web Distributed Authoring and Versioning (WebDAV).

SQL Server 2000

Microsoft SQL Server 2000 is the latest release of the SQL Server family. SQL Server was designed to provide all the tools needed to build powerful e-business applications. It includes built in support for XML, simplifying back office system integration and data transfer. Data can be retrieved directly as XML, and XML can be stored relationally.

This XML functionality allows Web developers, for example, to use technologies like XPath, URL queries, and XML updategrams, instead of needing in-depth knowledge of relational database programming. Similarly, database developers are not required to learn an object-oriented language or to understand all the facets of XML. They can provide XML access to an existing relational database with the FOR XML clause that returns XML data from a SELECT statement and the OPENXML T-SQL keyword.

Retrieving data as XML is important, as is the ability to store data efficiently as XML, maintaining the relationships and a hierarchy of data, while taking full advantage of the speed offered by a relational database. SQL Server 2000 can provide an XML View of relational data, as well as map XML data into relational tables.

OpenXML allows XML documents to be addressed with relational SQL syntax. OpenXML is a T-SQL keyword that provides an updateable rowset interface for in-memory XML documents. The records in the rowset can be stored in database tables, just as rowsets provided by tables and views. OpenXML can be used in SELECT and SELECT INTO statements wherever rowset providers (such as table, view, or OPENROWSET), can display.

The new services engineered into this latest version make SQL Server 2000 more scalable and easier to program. Analysis Services offers built-in data mining tools and storage options that make it possible to build and analyze data warehouses of any size. They have been expanded to allow OLAP cubes to be accessed and analyzed over the Web using HTTP, offering remote users (including suppliers and trading partners

outside the intranet) the ability to use SQL Server analysis tools. Significant upgrades to the querying tools make it easier for developers to build natural language queries.

SQL Server also includes new features that allow the workload to be partitioned for increased scalability. This is done by “horizontally” partitioning data across multiple servers, typically based on some data-driven attribute (such as the Last Name or Account ID). These servers manage the partitioned data together, but operate independently.

The partitioning of data is transparent to applications accessing the database. An application "sees" a full copy of all tables. All servers accept connections and process both queries and updates, distributing scans and updates as needed. The SQL Server 2000 query processor contains a number of enhancements so that partitioned views can be efficiently updated, plus increases distributed query performance.

SQL Server includes the ability to perform differential backups. It also incorporates many important security and availability features. SQL has been enhanced with security features built on the Windows security model (incorporating flexible role-based security for server, database, and application profiles), integrated tools for security auditing, and support for file and over-the-wire encryption. This augments operating system security and helps to fulfill legal requirements for storing sensitive data.

SQL Server 2000 is tightly integrated with many of the other members of the .Net Enterprise Server family. Microsoft Commerce Server 2000 provides services that include user profiling, product catalogs and Business Internet Analytics (BIA)—the analysis of customer Web click-stream data to make predictions about customer behavior and drive personalization. These services are built on SQL Server 2000.

Microsoft BizTalk Server 2000 also works with SQL Server. BizTalk Server provides the infrastructure and tools to enable e-commerce business communications. SQL Server 2000 and BizTalk Server support the same XML data-reduced schema, allowing, documents to be transmitted directly from SQL Server to BizTalk Server and vice versa.

BizTalk: The Concept

Many people think of BizTalk as simply another Microsoft product. Actually, BizTalk can also refer to the BizTalk Organization, the BizTalk Initiative, or BizTalk orchestration...or all four parts as a whole. In order to understand BizTalk completely, it is important to understand the sum of the parts, as well as the individual pieces.

The Organization

The BizTalk organization was formed several years ago. BizTalk is an independent organization, but it was founded at Microsoft's initiative. Its goal is and was to be a resource center for business and software communities for learning about and using

XML in documents and information-sharing over the Internet. BizTalk is an online resource rather than a consortium or a standards body.

This site is a library where XML and XSL schema can be located, managed and published. It contains information models and business processes supported by applications that support the BizTalk Framework.

For more information on the BizTalk organization, visit www.biztalk.org.

When two companies want to communicate electronically over the Internet, documents such as purchase orders, order acknowledgements, shipping documents, and so on, can be defined in XML. The problem is that the two companies that want to communicate need to agree on how the documents are to be expressed in XML, and this definition is a scheme. Here, people are actively encouraged to publish our own schema.

The Biztalk Initiative is made up of the Microsoft BizTalk Framework, various cross-industry investments, including the BizTalk.org business document library, as well as Microsoft BizTalk Server 2000. These investments are being made with industry standards groups, technology and service providers, as well as key global organizations. The first set of investments surround the BizTalk Framework, and this is a set of implementation guidelines that any organization can use to define and route business data using XML.

The BizTalk Framework itself is not a standard. XML is the standard. The goal of the BizTalk Framework is to accelerate the rapid adoption of XML. The BizTalk Framework implementation guidelines are documented and accessible on www.biztalk.org. Since the BizTalk Framework is 100 percent compliant with the World Wide Web Consortium (W3C) XML 1.0 standard, it is operating system, programming model, and programming language independent. BizTalk documents are being used by applications on many different platforms today. The framework provides the implementation guidelines for XML-based document routing.

The BizTalk Steering Committee drives the BizTalk Framework. The steering committee is made up of more than 20 industry standards organizations, large software companies, and multinational corporations that share the interest of promoting the use of XML to provide better interoperability between applications and processes.

BizTalk Orchestration

Every business process is comprised of well-defined steps. Each step started by the receipt of a message, in a certain pre-defined format and from a particular location. Each step ends also by the sending of a message, again in a certain format, to a

particular location. Automating these processes and messages is called orchestration. A design tool hosted in Microsoft Visio allows a business analyst to flowchart a particular business process. That diagram can then be mapped to physically implemented components in the .NET runtime environment. An XLANG messaging infrastructure will typically be used to deliver control information among the distributed elements of the system.

For Further Learning

This white paper is intended to assist IT professionals develop a full understanding of .NET and to help them learn how to begin building e-Business and e-Commerce applications using this new technology. Visit G. A. Sullivan's .NET portal site at www.gasTIX.net to get an introduction to an enterprise-scale sample application that illustrates the important concepts and techniques to keep in mind when building .NET applications. Full source code is made available for download to provide the optimal learning experience.

If you are interested in learning even more about the intricacies of building .NET applications, purchase the SAMS book entitled ".NET e-Business Architecture" (ISBN 0672322196). This book, written by a team of senior designers and developers at G. A. Sullivan, chronicles the best practices and lessons learned in building www.gasTIX.net.

Appendices

A. Helpful Web Sites

The following Web sites contain helpful information related to .NET application design and development. Content is updated often, new technologies are introduced, and service pack bulletins are announced. Check these sites often to stay current:

- http://www.microsoft.com/net
- http://msdn.microsoft.com/net
- http://www.gotdotnet.com
- http://www.asp.net
- http://www.gasTIX.net

B. Newsgroups

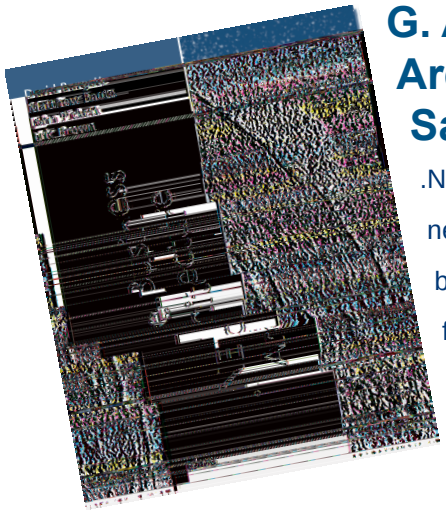
Several newsgroups focused on the .NET Framework and Visual Studio .NET are reachable by browsing the Microsoft Communities homepage:

- http://communities.microsoft.com

Additionally, the newsgroups at DevelopMentor are quite active:

- http://discuss.develop.com

.NET Leadership



G. A. Sullivan's .NET e-Business Architecture Available from Sams Publishing

.NET e-Business Architecture demonstrates the new Microsoft .NET architecture and toolset for building enterprise applications. This book features a sample application that demonstrates best practices for producing a complete e-Business application using Microsoft's .NET platform.

www.gasTIX.NET

G. A. Sullivan has created a .NET-focused Web portal, designed to provide practical information and hands-on tools enabling you to learn about implementing enterprise-class applications utilizing Microsoft's .NET platform. We have also created a live sample application to demonstrate how to develop a Web site leveraging .NET technologies:

- Web Services and Remoting
- ASP.NET
- C#
- Visual Basic.NET
- ADO.NET
- BizTalk
- SQL Server 2000

www.gasTIX.net

G.A. SULLIVAN *business*^{e}