DCOM Technical Overview

The following article is from the Microsoft website. The original document can be viewed at the Microsoft website here:

Microsoft's distributed COM (DCOM) extends the Component Object Model (COM) to support communication among objects on different computers-on a LAN, a WAN, or even the Internet. With DCOM, your application can be distributed at locations that make the most sense to your customer and to the application.

Because DCOM is a seamless evolution of COM, the world's leading component technology, you can leverage your existing investment in COM-based applications, components, tools, and knowledge to move into the world of standards-based distributed computing. As you do so, DCOM handles low-level details of network protocols so you can focus on your real business: providing great solutions to your customers.

On This Page

Introduction
Why write distributed applications?
The DCOM architecture
Components and Reuse
Location independence
Language neutrality
Connection Management
Scalability
Performance
Bandwidth and Latency
Security
Load Balancing
Fault Tolerance
Ease of Deployment
Protocol neutrality
Platform neutrality
Seamless Integration with other Internet protocols
Summary
References

Introduction

Microsoft's distributed COM (DCOM) extends the Component Object Model (COM) to support communication among objects on different computers—on a LAN, a WAN, or even the Internet. With DCOM, your application can be distributed at locations that make the most sense to your customer and to the application.

Because DCOM is a seamless evolution of COM, the world's leading component technology, you can take advantage of your existing investment in COM-based applications, components, tools, and knowledge to move into the world of standards-based distributed computing. As you

do so, DCOM handles low-level details of network protocols so you can focus on your real business: providing great solutions to your customers.

Why Should I Read This Paper?

If you are a CIO, solution architect, or application developer who wants to create state-of-the-art applications that scale equally well on an intranet, the Internet, and beyond, DCOM can help. This White Paper provides a high-level overview of how you can use DCOM to solve the hardest problems associated with distributed applications. Has any of your applications ever needed (or will need):

- to provide fault-tolerance in the case of hardware failures?
- to be installed in a small workgroup environment as well as a large corporate site?
- to be robust in the presence of network failures?
- to accommodate a wide variety of client machines with different capabilities or in different geographical areas?
- to be more efficient in terms of network load?

This paper will show you how DCOM can help you solve these and other problems in both existing applications and completely new designs.

How Should I Read This Paper?

This White Paper is the first in a series introducing COM and DCOM. This paper provides a high-level explanation of design issues that are addressed by DCOM:

- Location independence
- Connection management
- Scalability
- Performance
- Bandwidth and latency
- Security
- Load balancing
- Fault tolerance
- Ease of deployment
- Protocol neutrality
- Platform neutrality
- Seamless integration with other Internet protocols

Start with this paper, and then refer to others in the series for details on specific topics

If you are wondering how to apply DCOM to specific applications and solutions, take a look at [DCOM - Solutions in Action] to get an idea of what others have done. There is a good chance that you will be able to apply many of the ideas in these solutions to your own work.

If you really want to understand how everything works and fits together, see Most of the papers referenced in this whitepaper are available on the Technology Preview CD. URLs to documents which are available for download are listed below.

Take this guided tour of the inner and outer workings of DCOM and you'll see for yourself how DCOM realizes the promise of easy distributed computing without compromising flexibility, scalability, or robustness.

This paper introduces you to the world of distributed computing. It shows you some problem areas that you will, or might already have, run into and gives you some ideas about how you can solve these problems using the unique features of DCOM.

For fast readers, each major section of this White Paper contains a summarizing paragraph that is highlighted like this one. If you need to skip a section, be sure to read at least this summary.

Where Can I Get DCOM?

DCOM currently ships with Windows NT® 4.0 and will be available for Windows® 95 before the end of 1996. DCOM for the Apple® Macintosh® will be available from Microsoft in early 1997. In addition, DCOM implementations on all major UNIX platforms, including a reference implementation in source code form, will become available in early 1997. COM and DCOM are no longer proprietary to Microsoft, but are managed by the independent ActiveX™ Consortium.

Why write distributed applications?

Distributing an application is not an end in itself. Distributed applications introduce a whole new kind of design and deployment issues. For this added complexity to be worthwhile, there has to be a significant payback.

Some applications are inherently distributed: multiuser games, chat and teleconferencing applications are examples of such applications. For these, the benefits of a robust infrastructure for distributed computing are obvious.

Many other applications are also distributed, in the sense that they have at least two components running on different machines. But because these applications were not designed to be distributed, they are limited in scalability and ease of deployment. Any kind of workflow or groupware application, most client/server applications, and even some desktop productivity applications essentially control the way their users communicate and cooperate. Thinking of these applications as distributed applications and running the right components in the right places benefits the user and optimizes the use of network and computer resources. The application designed with distribution in mind can accommodate different clients with different capabilities by running components on the client side when possible and running them on the server side when necessary.

Designing applications for distribution gives the system manager a great deal of flexibility in deployment.

Distributed applications are also much more scalable than their monolithic counterparts. If all the logic of a complex application is contained in a single module, there is only one way to increase the throughput without tuning the application itself: faster hardware. Today's servers and operating systems scale very well but it is often cheaper to buy another identical machine than to upgrade to a server that is twice as fast. With a properly designed distributed application, a single server can start out running all the components. When the load increases, some of the components can be deployed to additional lower-cost machines.

The DCOM architecture

DCOM is an extension of the Component Object Model (COM). COM defines how components and their clients interact. This interaction is defined such that the client and the component can connect without the need of any intermediary system component. The client calls methods in the component without any overhead whatsoever. Figure 1 illustrates this in the notation of the Component Object Model:

Figure 1: - COM Components in the same process

In today's operating systems, processes are shielded from each other. A client that needs to communicate with a component in another process cannot call the component directly, but has to use some form of inter-process communication provided by the operating system. COM provides this communication in a completely transparent fashion: it intercepts calls from the client and forwards them to the component in another process. Figure 2 illustrates how the COM/DCOM run-time libraries provide the link between client and component.

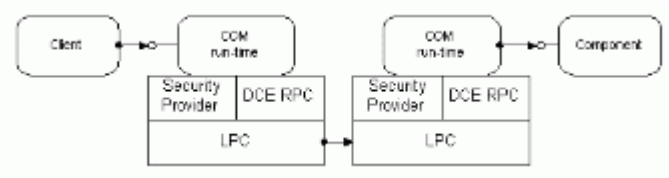

Figure 2: - COM Components in different processes

When client and component reside on different machines, DCOM simply replaces the local Inter-process communication with a network protocol. Neither the client nor the component are aware that the wire that connects them has just become a little longer.

Figure 3 shows the overall DCOM architecture: The COM run-time provides object-oriented services to clients and components and uses RPC and the security provider to generate standard network packets that conform to the DCOM wire-protocol standard.

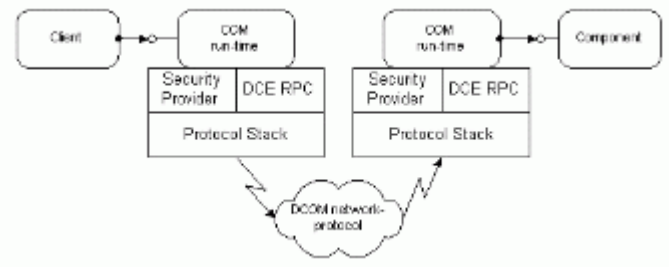

Figure 3: - DCOM: COM Components on different machines

Components and Reuse

Most distributed applications are not developed from scratch and in a vacuum. Existing hardware infrastructure, existing software, existing components as well as existing tools need to be integrated and leveraged to reduce development and deployment time and cost. DCOM directly and transparently takes advantage of any existing investment in COM components and tools. A huge market for off-the-shelf components makes it possible to reduce development time by integrating standardized solutions into a custom application. Many developers are familiar with COM and can easily apply their knowledge to DCOM-based distributed applications.

Any component that is developed as part of a distributed application is a candidate for future reuse. Organizing the development process around the component paradigm, lets you continuously raise the level of functionality in new applications and reduce time-to-market by building on previous work.

Designing for COM and DCOM assures that your components are useful now and in the future.

Location independence

When you begin to implement a distributed application on a real network, several conflicting design constraints become apparent:

- Components that interact more should be "closer" to each other.
- Some components can only be run on specific machines or at specific locations.
- Smaller components increase flexibility of deployment, but they also increase network traffic.
- Larger components reduce network traffic, but they also reduce flexibility of deployment..

With DCOM these critical design constraints are fairly easy to work around, because the details of deployment are not specified in the source code. DCOM completely hides the location of a component, whether it is in the same process as the client or on a machine halfway around the world. In all cases, the way the client connects to a component and calls the component's methods is identical. Not only does DCOM require no changes to the source code, it does not even require that the program be recompiled. A simple reconfiguration changes the way that components connect to each other.

DCOM's location independence greatly simplifies the task of distributing application components for optimum overall performance. Suppose, for example, that certain components must be placed on a specific machine or at a specific location. If the application has numerous small components, you can reduce network loading by deploying them on the same LAN segment, on the same machine, or even in the same process. If the application is composed of a smaller number of large components, network loading is less of a problem, so you can put them on the fastest machines available, wherever those machines are.

Figure 4 shows how the same "validation component" can be deployed on the client machine, when network-bandwidth between "client" machine and "Middle-tier" machine is sufficient, and on the server machine, when the client is accessing the application through a slow network link.

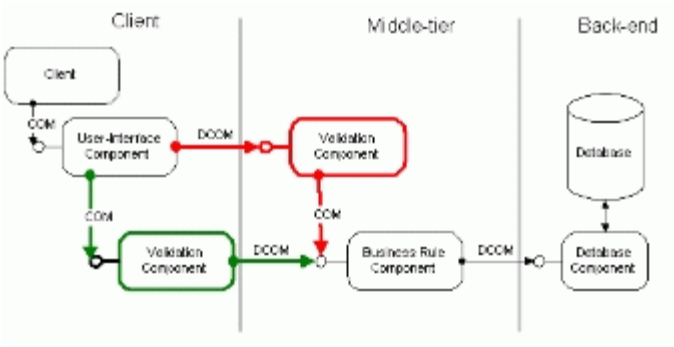

Figure 4: - Location Independence

With DCOM's location independence, the application can combine related components into machines that are "close" to each other onto a single machine or even into the same process. Even if a larger number of small components implement the functionality of a bigger logical module, they can still interact efficiently among each other. Components can run on the machine where it makes most sense: user interface and validation on or close to the client, database-intensive business rules on the server close to the database.

Language neutrality

A common issue during the design and implementation of a distributed application is the choice of the language or tool for a given component. Language choice is typically a trade-off between development cost, available expertise, and performance. As an extension of COM, DCOM is completely language-independent. Virtually any language can be used to create COM components, and those components can be used from even more languages and tools. Java, Microsoft® Visual C++®, Microsoft Visual Basic®, Delphi, PowerBuilder, and Micro Focus COBOL all interact well with DCOM.

With DCOM's language independence, application developers can choose the tools and languages that they are most familiar with. Language independence also enables rapid prototyping: components can be first developed in a higher-level language, such as Microsoft Visual Basic, and later reimplemented in a different language, such as C++ or Java, that can better takes advantage of advanced features such as DCOM's free-threading/multithreading and thread-pooling.

Connection Management

Network connections are inherently more fragile than connections inside a machine. Components in a distributed application need to be notified if a client is not active anymore, even—or especially—in the case of a network or hardware failure.

DCOM manages connections to components that are dedicated to a single client, as well as components that are shared by multiple clients, by maintaining a reference count on each component. When a client establishes a connection to a component, DCOM increments the component's reference count. When the client releases its connection, DCOM decrements the component's reference count. If the count reaches zero, the component can free itself.

DCOM uses an efficient pinging protocol (see section 0 "Shared Connection Management Between Applications") to detect if clients are still active. Client machines send a periodic message. DCOM considers a connection as broken if more than three ping periods pass without the component receiving a ping message. If the connection is broken, DCOM decrements the reference count and releases the component if the count has reached zero. From the point of view of the component, both the benign case of a client disconnecting and the fatal case of a network or client machine crash are handled by the same reference counting mechanism. Applications can use this distributed garbage collection mechanism for free.

In many cases, the flow of information between a component and its clients is not unidirectional: the component needs to initiate some operation on the client side, such as a notification that a lengthy process has finished, the update of data the user is viewing (news ticker or stock ticker), or the next message in a collaborative environment like teleconferencing or a multiuser game. Many protocols make it difficult to implement this kind of symmetric communication. With DCOM, any component can be both a provider and a consumer of functionality. The same mechanism and features manage communication in both directions, making it easy to implement peer-to-peer communication, as well as client/ server interactions.

DCOM provides a robust distributed garbage collection mechanism that is completely transparent to the application. DCOM is a inherently symmetric network protocol and programming model. Not only does it offer the traditional unidirectional client-server interaction, but it also provides rich, interactive communication between clients and servers and among peers.

Scalability

A critical factor for a distributed application is its ability to grow with the number of users, the amount of data, and the required functionality. The application should be small and fast when the demands are minimal, but it should be able to handle additional demands without sacrificing performance or reliability. DCOM provides a number of features that enhance your application's scalability.

Symmetric Multiprocessing (SMP)

DCOM takes advantage of Windows NT support for multiprocessing. For applications that use a free-threading model, DCOM manages a thread pool for incoming requests. On multiprocessor machines, this thread pool is optimized to the number of available processors: too many threads result in too much context switching, while too few threads can leave some processors idle. DCOM shields the developer from the details of thread management and delivers the optimal performance, that only costly hand-coding of a thread pool manager could provide.

DCOM applications can easily scale from small single processor machines to huge multiprocessor systems, by seamlessly taking advantage of Windows NT support for symmetric multiprocessing.

Flexible Deployment

As the load on an application grows, not even the fastest multiprocessor machine may be able to accommodate demand, even if your budget can accommodate such a machine. DCOM's location

independence makes it easy to distribute components over other computers, offering an easier and less expensive route to scalability.

Redeployment is easiest for stateless components or for those that do not share their state with other components. For components such as these, it is possible to run multiple copies on different machines. The user load can be evenly distributed among the machines, or criteria like machine capacity or even current load can be take into consideration. With DCOM, it is easy to change the way clients connect to components and components connect to each other. The same components can be dynamically redeployed, without any rework or even recompilation. All that is necessary is to update the registry, file system, or database where the location of each component is stored.

Example: An organization with offices in multiple locations, such as New York, London , San Francisco, and Sydney, Australia, can install the components in its servers. Two hundred users simultaneously access 50 components in a server with the expected performance. As new business applications are delivered to users, applications that use some existing business components and some newer ones, the load on the server grows to 600 users, and the number of business components grows to 70. With these additional applications and users, response times become unacceptable during peak hours. The administrator adds a second server and redeploys 30 of the components exclusively on the new machine. Twenty components remain exclusively on the old server, while the remaining 20 are run on both machines.

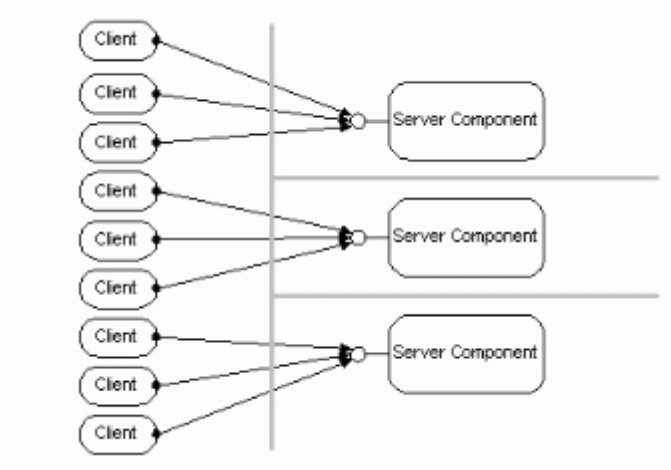

Figure 5: - Parallel Deployment

Most real world applications have one or more critical components that are involved in most of the operations. These can be database components or business rule components that need to be accessed serially to enforce "first-come, first served" policies. These kind of components cannot be duplicated, since their sole purpose is to provide a single synchronization point among all users of the application. To improve the overall performance of a distributed application, these "bottleneck" components have to be deployed onto a dedicated, powerful server. Again, DCOM helps by letting you isolate these critical components early in the design process, deploying multiple components on a single machine initially and moving the critical components to dedicated machines later. As explained in Section 0 of this paper, no redesign or even recompilation of the components is needed.

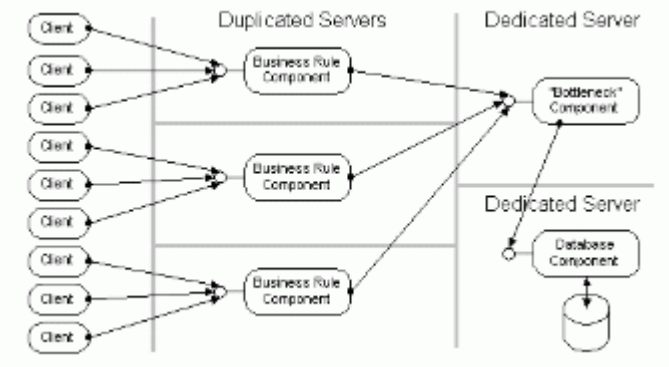

Figure 6: - Isolating Critical Components

For these critical bottleneck components, DCOM can make the overall task go more quickly. Such bottlenecks are usually part of a processing sequence, such as buy or sell orders in an electronic trading system: requests must be processed in the order they are received (first come, first served). One solution is to break the task into smaller components and deploy each component on a different machine. The effect is similar to pipelining as used in modern microprocessors: the first request comes in, the first component processes it (does, for example, consistency checking) and passes the request on to the next component (which might, for example, update the database). As soon as the first component passes the request on to the second component, it is ready to process the next request. In effect, there are two machines working in parallel on multiple requests, while the order in which requests are processed is guaranteed. The same approach is possible using DCOM on a single machine: multiple components can run on different threads or in different processes. This approach simplifies scalability later, when the threads can be distributed on a multiprocessor machine or the processes can be deployed on different machines.

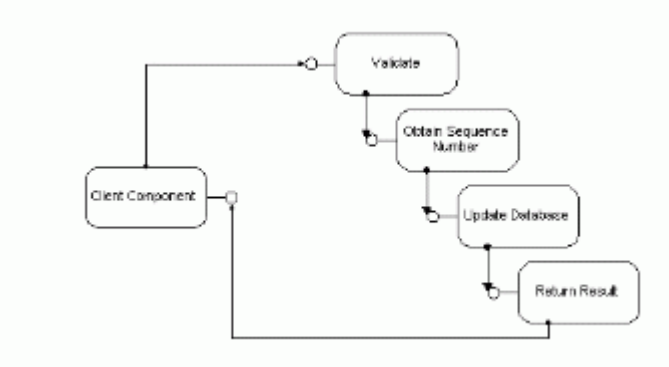

Figure 7: - Pipelining

DCOM's location-independent programming model makes it easy to change deployment schemes as the application grows: a single server machine can host all the components initially, connecting them as very efficient in-process servers. Effectively, the application behaves as a highly tuned monolithic application. As demand grows, other machines can be added, and the components can be distributed among those machines without any code changes.

Evolving Functionality: Versioning

Besides scaling with the number of users or the number of transactions, applications also need to scale as new features are required . Over time, new tasks need to be incorporated and existing ones modified. In the conventional approach, either clients and components have to be updated simultaneously or the old component has to be retained until all clients have upgraded—an undertaking that can become a major administrative burden when a significant number of geographically dispersed sites or users is involved.

DCOM provides flexible evolutionary mechanisms for clients and components. With COM and DCOM, clients can dynamically query the functionality of the component. Instead of exposing its functionality as a single, monolithic group of methods and properties, a COM component can appear differently to different clients. A client that uses a certain feature needs access only to the methods and properties it uses. Clients can also use more than one feature of a component simultaneously. If other features are added to the component, they do not affect an older client that is not aware of them.

Being able to structure components this way, enables a new kind of evolution: the initial component exposes a core set of features as COM interfaces, on which every client can count. As the component acquires new features, most (often even all) of these existing interfaces will still be necessary; and new functions and properties appear in additional interfaces without changing the original interfaces at all. Old clients still access the core set of interfaces as if nothing had changed. New clients can test for the presence of the new interfaces and use them when available, or they can degrade gracefully to the old interfaces.

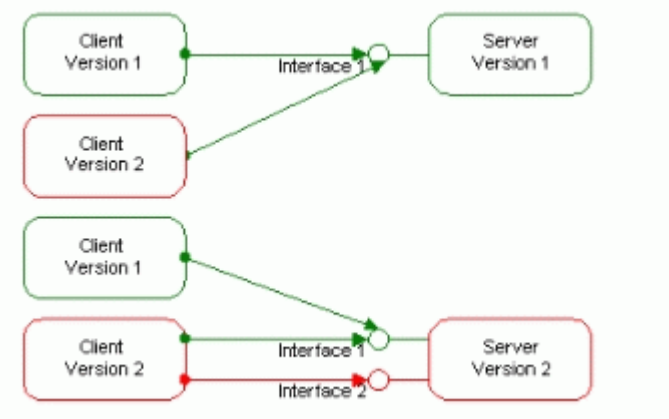

Figure 8: - Robust Versioning

Because functionality is grouped into interfaces in the DCOM programming model, you can design new clients that run with old servers, new servers that run with old clients, or mix and match to suit your needs and programming resources. With conventional object models, even a slight change to a method fundamentally changes the contract between the client and the component. In some models, it is possible to add new methods to the end of the list of methods, but there is no way to safely test for the new methods on old components. From the network's perspective, things become even more complicated: encoding and wire-representation typically depend on the order of the methods and parameters. Adding or changing methods and parameters also changes the network protocol significantly. DCOM handles all these problems with a single, elegant, unified approach for both the object model and the network protocol.

Performance

Scalability is not much of a benefit if the initial performance is not satisfactory. It is always good to know that more and better hardware can take an application to its next evolutionary step, but what about the entry-level requirements? Don't all these high-end scalability features come at a price? Doesn't supporting every language from COBOL to Assembler necessarily compromise performance? Doesn't the ability to run a component on the other side of the world preclude running it efficiently in the same process as the client?

In COM and DCOM, the client never sees the server object itself, but the client is never separated from the server by a system component unless it's absolutely necessary. This transparency is achieved by a strikingly simple idea: the only way a client can talk to the component is through method calls. The client obtains the addresses of these methods from a simple table of method addresses (a "vtable"). When the client wants to call a method on a component, it obtains the method's address and calls it. The only overhead incurred by the COM programming model over a traditional C or Assembler function call is the simple lookup of the method's address (indirect function call *vs.* direct function call). If the component is an in-process component running on the same thread as the client, the method call arrives directly at the component. No COM or system code is involved; COM only defines the standard for laying out the method address table.

What happens when the client and the component are actually not as close—on another thread, in another process, or on another machine at the other side of the world? COM places its own remote procedure call (RPC)-infrastructure code into the vtable and then packages each method call into a standard buffer representation, which it sends to the component's side, unpacks it, and reissues the original method call: COM provides an object-oriented RPC mechanism.

How fast is this RPC mechanism? There are different performance metrics to consider:

- How fast is an "empty" method call?
- How fast are "real world" method calls that send and return data?
- How fast is a network round trip?

The table below shows some real-world performance numbers for COM and DCOM to give an idea of the relative performance of DCOM compared to other protocols.

Parameter Size	**4 bytes**		**50 bytes**	
	calls / sec	**ms / call**	**calls / sec**	**ms / call**
"Pentium®„" in-process	3,224,816	0.00031	3,277,973	0.00031
"Alpha™," in-process	2,801,630	0.00036	2,834,269	0.00035
"Pentium," cross-process	2,377	0.42	2,023	0.49
"Alpha," cross-process	1,925	0.52	1634	0.61
"Alpha," to Pentium remote	376	2.7	306	3.27

* These informal numbers were obtained on the author's Dell OptiPlex XM 5120 (120 MHz Pentium, 32MB RAM) and a small DEC™ Alpha-based RISC-machine (200 MHz, 32MB RAM). Both machines were running the release version of Windows NT 4.0 (Build 1381). DCOM was using UDP over Intel® EtherExpress PRO network cards (10 Mbps) on the Microsoft corporate network under a normal load. The COM Performance Sample - available in the Windows NT 4.0 Win32 SDK - can be used to obtain similar numbers with other configurations.

The first two columns represent an "empty" method call (passing in and returning a 4-byte integer). The last two columns can be considered a "real world" COM method call (50 bytes of parameters).

The table shows how in-process components obtain zero-overhead performance (rows 1 and 2).

Cross-process calls (rows 3 and 4) require the parameters to be stored into a buffer and sent to the other process. A performance of roughly 2000 calls per second on standard desktop hardware, satisfies most performance requirements. All local calls are completely bound by processor speed (and to some extent by available memory) and scale well on multi-processor machines.

Remote calls (rows 5 and 6) are primarily network bound and indicate approximately 35% overhead of DCOM over raw TCP/IP performance (2 ms roundtrip time for TCP/IP).

Microsoft will soon provide formal DCOM performance numbers on a wide range of platforms, that show DCOM's ability to scale with the number of clients and with the number of processors on the server.

These informal - but reproducible - performance numbers indicate an overhead of approximately 35% of DCOM over raw TCP/IP for empty calls. This ratio decreases further as the server performs actual processing. If the server requires 1 ms - for example to update a database - the ratio decreases to 23% and to 17% if the server requires 2 ms.

The overall performance and scalability advantages of DCOM can only be reached by implementing sophisticated thread-pool managers and pinging protocols. Most distributed applications will not want or need to incur this significant investment for obtaining minor performance gains, while sacrificing the convenience of the standardized DCOM wire-protocol and programming model.

Bandwidth and Latency

Distributed applications take advantage of a network to tie components together. In theory, DCOM completely hides the fact that components are running on different computers. In practice however, applications need to consider the two primary constraints of a network connection:

- Bandwidth: the size of the parameters passed into a method call directly affects the time it takes to complete the call.
- Latency: the physical distance and the number of network elements involved (such as routers and communication lines) delay even the smallest data packet significantly. In the case of a global network like the Internet, these delays can be on the order of seconds.

How does DCOM help applications to deal with these constraints? DCOM itself minimizes network round trips wherever possible to avoid the impact of network latency. DCOM's preferred transport protocol is the connectionless UDP subset of the TCP/IP protocol suite: The connectionless nature of this protocol allows DCOM to perform several optimizations by merging many low-level acknowledge packages with actual data and pinging messages. Even running over connection-oriented protocols, DCOM still offers significant advantages over application-specific custom protocols.

Shared Connection Management Between Applications

Most application level protocols require some kind of lifetime management. The component needs to get notified when a client machine suffers a catastrophic hardware failure or the network connection between client and component breaks for an extended period of time.

A common approach to this problem is to send keep-alive message at periodic intervals (pinging). If the server does not receive a ping message for a specified time, it declares the client "dead."

DCOM uses a per machine keep-alive message. Even if the client machine uses 100 components on a server machine, a single ping message keeps all the clients connections alive. In addition to consolidating all the ping messages, DCOM minimizes the size of these ping messages by using delta pinging. Instead of sending 100 client identifiers, it creates meta-identifiers that represent all 100 references. If the set of references changes, only the delta between the two reference sets is transmitted. Finally, DCOM piggybacks the ping message onto regular messages. Only if the entire client machine is idle with respect to a given server machine does it send periodic ping messages (at a 2-minute interval).

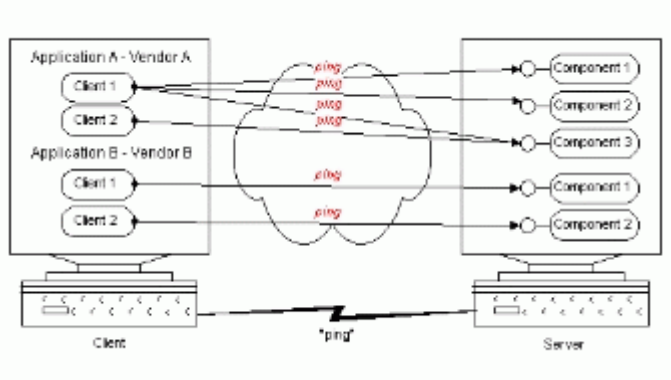

Figure 9: - Consolidated lifetime management

DCOM allows different applications (even from different vendors) to share a single, optimized lifetime management and network failure detection protocol, reducing bandwidth significantly. If 100 different applications with 100 different custom protocols are running on a server, this server would normally receive one ping message for each of those applications from each of the connected clients. Only if these protocols somehow coordinate their pinging strategies can the overall network overhead be reduced. DCOM automatically provides this coordination among arbitrary COM-based custom protocols.

Optimize Network Round-Trips

A common problem in designing distributed applications is an excessive number of network round-trips between components on different machines. On the Internet, each of these round-trips incurs a delay of typically 1 second, often significantly more. Even over a fast local network, round-trip times are typically measured in milliseconds—orders of magnitude above the cost of local operations.

A common technique for reducing the number of network round trips is to bundle multiple method calls into a single method invocation (batching or boxcarring). DCOM uses this technique extensively for tasks such as connecting to an object or creating a new object and querying its functionality (see section 0). The disadvantage of this technique for general components is that the programming model changes significantly between the local and the remote case.

Example: A database component provides a method for enumerating the results of a query either row by row or several rows at a time. In the local case, a developer can simply use this method to add the rows one by one to a list box. In the remote case, this approach would incur a network round trip for each row enumerated. Using the method in a batched fashion requires the developer to allocate a buffer large enough to hold all the rows in the query and retrieve them in one call, then adding them to the list box one by one. Because the programming model has changed significantly, the developer has to make design compromises so the application will work efficiently in a distributed environment.

DCOM makes it easy for component designers to perform batching without requiring the clients to use a batching-style API. DCOM's marshaling mechanism lets the component inject code on the client side, called a "proxy object," that can intercept multiple method calls and bundle them into a single remote procedure call:

Example: The developer of the previous example continues to enumerate the methods one by one, since that is the way the application's logic requires it. (The list box API requires this.) However, the first call for enumerating the query result arrives in the application-specific proxy object, which retrieves all the rows (or a reasonable "batch" of rows) and caches them in the proxy object. Subsequent calls then come from this cache without additional network round trips. The developer continues with a simple programming model, yet the overall application is optimized.

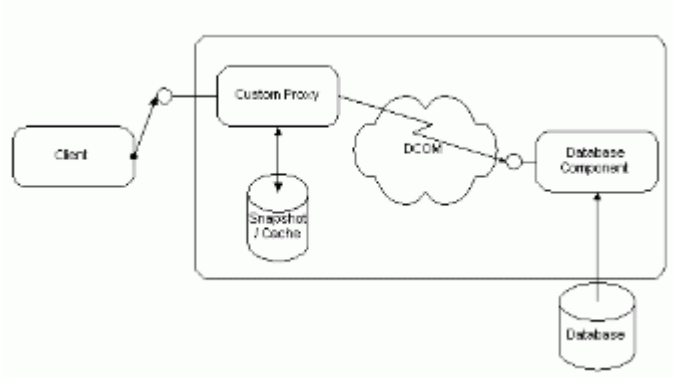

Figure 10: - The component model: Client-side caching

DCOM also allows efficient referrals from one component to the other. If a component holds a reference to another component on a separate machine, it can pass this reference to a client

running on a third machine (refer the client to another component running on another machine). When the client uses this reference, it communicates directly with the second component. DCOM short-circuits these references and lets the original component and machine get out of the picture entirely. This enables custom directory services that can return references to a wide range of remote components:

Example 1: A chess application can allow players who are waiting for a partner to register themselves with a chess directory service. Other players can browse or query the list of waiting players. When a player chooses a partner, the chess directory service returns a reference to the partner's client component. DCOM automatically connects the two players; the directory service is not involved in any further transactions.

Example 2: A "broker" component keeps track of a pool of 20 server machines running identical components. It measures the current load on the servers and detects when servers are added or removed. When a client needs a component, it connects to the "broker" component, which returns a reference to a component on the server with the lowest load. DCOM automatically connects the client to the server; the "broker" component then gets completely out of the way.

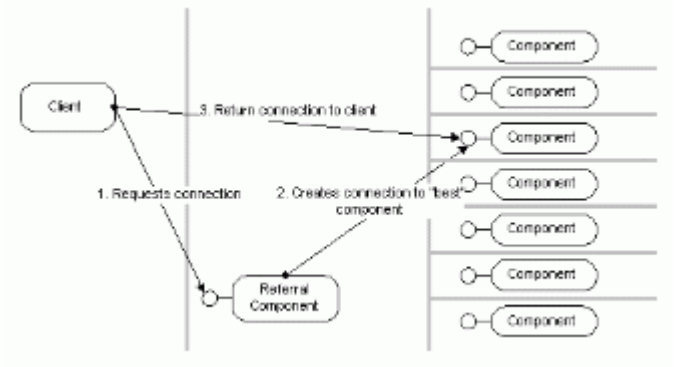

Figure 11: - Referral

If necessary, DCOM even allows components to plug in arbitrary custom protocols that use means outside of the DCOM mechanism. The component can use custom marshaling to inject an arbitrary proxy object into the client process, which can then use any arbitrary protocol to talk back to the component.

Example 1: A server-side component might be using an ODBC connection to talk to an SQL Server database. When a client retrieves this object, it may be more efficient to have the client machine communicate directly with the SQL Server™ database (using ODBC) instead of using DCOM to communicate with the server machine, which in turn communicates with the SQL Server database. With DCOM's custom marshaling, the database component can basically copy itself onto the client machine and connect itself to the SQL Server without the client ever noticing that it is connected not to the database component on the server anymore, but to a local copy of the same database component.

Example 2: A trading system needs two kinds of communication mechanisms: a secure, authenticated channel from clients to a central system, which is used for placing and revoking orders, and a distribution channel, which disseminates order information simultaneously to all connected clients. While the client/server channel is handled efficiently and easily using today's DCOM secure and synchronous connections, the broadcast channel might require a more

sophisticated mechanism using multicast technologies to accommodate large numbers of listeners. DCOM allows this custom protocol ("reliable broadcast") to be plugged seamlessly into the application architecture: a data sink component can encapsulate this protocol and make it completely transparent to both client and server. For small installations with few users, standard DCOM point-to-point protocols can be used, while larger customer sites would use the sophisticated custom broadcast protocol. If DCOM provides a standard multicast transport in the future, the application can migrate seamlessly to the new protocol.

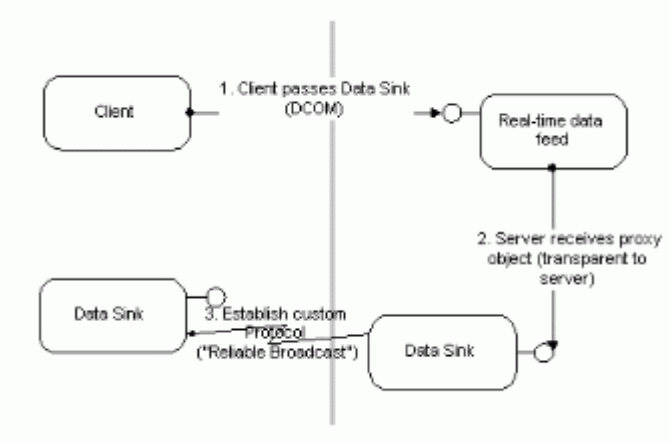

Figure 12: - Replacing DCOM with custom protocols

DCOM provides a multitude of ways to "tweak" the actual network protocol and network traffic without changing the way that clients perceive the component: client-side caching, referrals and replacing the network transport when necessary are but a few techniques that are possible.

Security

Using the network for distributing an application is challenging not only because of the physical limitations of bandwidth and latency. It also raises new issues related to security between and among clients and components. Since many operations are now physically accessible by anyone with access to the network, access to these operations has to be restricted at a higher level.

Without security support from the distributed development platform, each application would be forced to implement its own security mechanisms. A typical mechanism would involve passing some kind of username and password (or a public key)—usually encrypted—to some kind of logon method. The application would validate these credentials against a user database or directory and return some dynamic identifier for use in future method calls. On each subsequent call to a secure method, the clients would have to pass this security identifier. Each application would have to store and manage a list of usernames and passwords, protect the user directory against unauthorized access, and manage changes to passwords, as well as dealing with the security hazard of sending passwords over the network.

A distributed platform must thus provide a security framework to safely distinguish different clients or different groups of clients so that the system or the application has a way of knowing who is trying to perform an operation on a component. DCOM uses the extensible security framework provided by Windows NT. Windows NT provides a solid set of built-in security providers that support multiple identification and authentication mechanisms, from traditional trusted-domain security models to noncentrally managed, massively scaling public-key security

mechanisms. A central part of the security framework is a user directory, which stores the necessary information to validate a user's credentials (user name, password, public key). Most DCOM implementations on non-Windows NT platforms provide a similar or identical extensibility mechanism to use whatever kind of security providers is available on that platform. Most UNIX-implementations of DCOM will include a Windows NT-compatible security provider.

Before looking more closely at these Windows NT security and directory providers, let's take a look at how DCOM uses this general security framework to make building secure applications easier.

Security by Configuration

DCOM can make distributed applications secure without any security-specific coding or design in either the client or the component. Just as the DCOM programming model hides a component's location, it also hides the security requirements of a component. The same (existing or off-the-shelf) binary code that works in a single-machine environment, where security may be of no concern, can be used in a distributed environment in a secure fashion.

DCOM achieves this security transparency by letting developers and administrators configure the security settings for each component. Just as the Windows NT File System lets administrators set access control lists (ACLs) for files and directories, DCOM stores Access Control Lists for components. These lists simply indicate which users or groups of users have the right to access a component of a certain class. These lists can easily be configured using the DCOM configuration tool (DCOMCNFG) or programmatically using the Windows NT registry and Win32® security functions.

Whenever a client calls a method or creates an instance of a component, DCOM obtains the client's current username associated with the current process (actually the current thread of execution). Windows NT guarantees that this user credential is authentic. DCOM then passes the username to the machine or process where the component is running. DCOM on the component's machine then validates the username again using whatever authentication mechanism is configured and checks the access control list for the component (actually for the first component run in the process containing the component. For details, see the "DCOM Architecture" White Paper.) If the client's username is not included in this list (either directly or indirectly as a member of a group of users), DCOM simply rejects the call before the component is ever involved. This default security mechanism is completely transparent to both the client and the component and is highly optimized. It is based on the Windows NT security framework, which is probably one of the most heavily used (and optimized!) parts of the Windows NT operating system: on each and every access to a file or even to a thread-synchronization primitive like an event or semaphore, Windows NT performs an identical access check. The fact that Windows NT can still compete with and beat the performance of competing operating systems and network operating systems shows how efficient this security mechanism is.

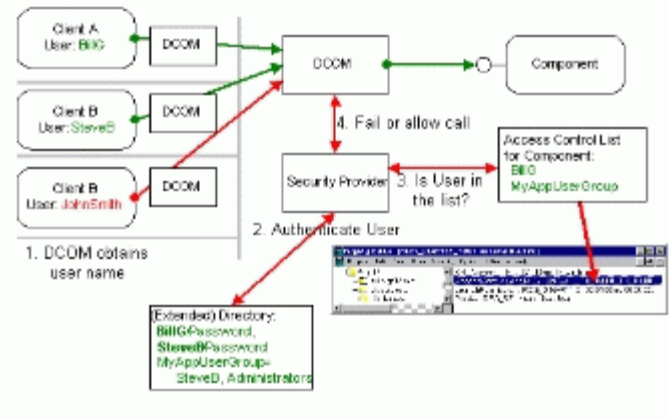

Figure 13: - Security by Configuration

DCOM provides an extremely efficient default security mechanism that lets developers write secure distributed applications without having to worry about security at all. Any security provider supported by Windows NT can be used with DCOM's security mechanism.

Programmatic Control Over Security

For some applications, a single component-wide access control list is not sufficient. Some methods in a component may be accessible only to certain users.

Example: An accounting business component may have a method for registering new transactions and another method for retrieving existing transactions. Only members of the accounting department (user group "Accounting") should be able to add new transactions, while only members of upper management (user group "Upper Management") should be able to view the transactions.

As indicated in the previous section, applications can always implement their own security by managing their own user database and security credentials. However, working from a standardized security framework provides many benefits to end users. Without a security framework , users have to remember and manage logon credentials for each application they are using. Developers have to be aware of security in each and every component of their applications.

DCOM simplifies customizing security to the needs of specific components and applications, providing extreme flexibility while incorporating any security standard supported by Windows NT. See the following section for details.

How can an application use DCOM security to implement the selective security required in the examples above? When a method call comes in, the component asks DCOM to impersonate the client. After this, the called thread can perform only those operations on secured objects, that the client is permitted to perform. The component can then try to access a secured object, such as a registry key, that has an Access Control List on it. If this access fails, the client was not contained in the ACL, and the component rejects the method call. By choosing different registry keys according to the method that is being called, the component can provide selective security in a very easy, yet flexible and efficient way.

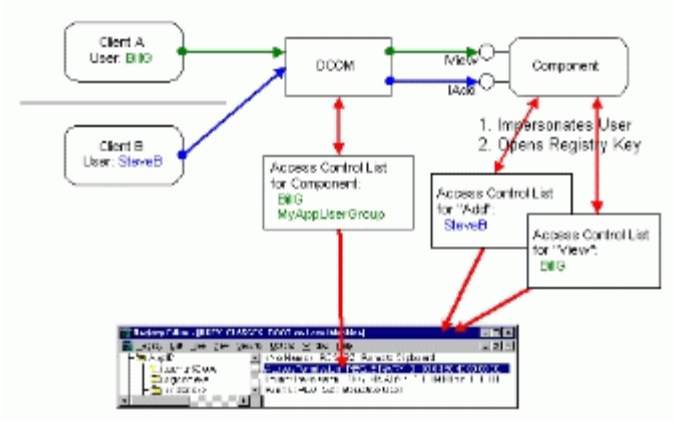

Figure 14: - Per interface security using registry keys

Components can also simply obtain the authenticated username of the client and use it to look up permissions or policies in their own database. This strategy employs the authentication mechanism of the Windows NT security framework (password/public key, encrypted passwords on the wire, etc.). The application does not have to worry about storing passwords or other sensitive information. The next version of Windows NT will provide an extended directory service that allows applications to store custom data inside the Windows NT user database.

DCOM provides even more flexibility. Components can require different levels of encryption and different levels of authentication, while clients can prevent components from using their credentials when impersonating. For more details on DCOM's advanced security infrastructure, see the "DCOM Architecture" White Paper.

Security on the Internet

There are two basic challenges facing applications designed to work over the Internet.

- The number of users can be orders of magnitude higher than in even the largest company.
- End users want to use the same key or password for all of the applications they are using, even if they are run by different companies. The application or the security framework on the provider side cannot store the private key of the user.

How can DCOM's flexible security architecture help applications to deal with these problems? DCOM uses the security framework provided by Windows NT. (See [Security] for more details.) The Windows NT security architecture supports multiple security providers, including:

- Windows NT NTLM authentication protocol, which is used by Windows NT 4.0 and previous versions of Windows NT.
- The Kerberos Version 5 authentication protocol, which replaces NTLM as the primary security protocol for access to resources within or across Windows NT domains.
- Distributed password authentication (DPA), the shared secret authentication protocol used by some of the largest Internet membership organizations, such as MSN™ and CompuServe.
- Secure channel security services, which implement the SSL/PCT protocols in Windows NT 4.0. The next generation of Windows NT security has enhanced support for public-key protocols that support SSL 3.0 client authentication.
- A DCE-compliant security provider, available as a third-party add-on to Windows NT.

All of these providers work over standard Internet protocols and have different advantages and disadvantages. The NTLM security provider and the Kerberos-based provider replacing it in Windows NT 5.0 are private key based protocols. They are extremely efficient and secure in centrally administered environments or a collection of Windows NT Server-based domains with mutual or unilateral trust-relations. Commercial implementations of NTLM security providers are available for all major Unix platforms (such as AT&T's "Advanced Server for Unix Systems").

With the Windows NT 4.0 directory service, multimaster domains scale well up to approximately 100,000 users. With the extended directory service in Windows NT 5.0, a single Windows NT domain controller can scale to approximately a million users. By combining multiple domain controllers into the Windows NT 5.0 directory tree, the number of users it is possible to support in a single domain is practically unlimited.

The Windows NT 5.0 Kerberos-based security provider allows even more advanced security concepts, such as control over what components can do while impersonating clients. This security provider also requires fewer resources for performing authentication than the NTLM security provider. See [Security] for more details.

Windows NT 5.0 will also include a public-key based security provider. This provider makes it possible to decentralize management of security credential with any Windows NT application, including DCOM-based applications. Authentication with public keys is less efficient than it is with private keys, but it allows authentication without storing the client's private credentials.

A wide range of fundamentally different security providers (private key, public-key) can be used by DCOM-based distributed applications without requiring any change to even advanced, security sensitive applications. The Windows NT security framework makes writing scalable and secure applications easy, without sacrificing flexibility and performance.

Load Balancing

The more successful a distributed application, the higher the load that the growing number of users places on all components of the application. Often, even the computing power of the fastest hardware is not enough to keep up with the user demand.

An inevitable option at this point is the distribution of the load among multiple server machines. Section 0, "Scalability," mentions briefly how DCOM facilitates different techniques of load balancing: parallel deployment, isolating critical components, and pipelining of sequential processes.

"Load balancing" is a widely used term that describes a whole set of related techniques. DCOM does not transparently provide load balancing in all its different meanings, but it does make it easy to implement different types of load balancing.

Static Load Balancing

One method of load balancing is to permanently assign certain users to certain servers running the same application. Because these assignments to not change with conditions on the network or other factors, this method is called static load balancing.

DCOM-based applications can be easily configured to use specific servers by changing a registry entry. Custom configuration tools can use the Win32 remote registry functions to change these settings on each client. With Windows NT 5.0, DCOM will use the extended directory service for implementing a distributed class store, which will make it possible to centralize these configuration changes.

With Windows NT 4.0, applications can use some simple techniques to achieve the same results. A basic approach is to store the server name in some well-known central location, such as a database or a small file. The client component simply retrieves the server name whenever it needs to connect to the server. Changing the database or the file contents changes all clients or arbitrary groups of clients simultaneously.

A much more flexible approach uses a dedicated referral component. This component resides on a well-known server machine. Client components connect first to this component, requesting a reference to the service they require. The referral component can use DCOM's security mechanisms to identify the requesting user and choose the server depending on who is making the request. Instead of just returning the name of the server, the referral component can actually establish a connection to this server and return it directly to the client. DCOM then transparently connects the client directly to the server; and the referral component gets completely out of the way. It is even possible to completely hide this mechanism from the client by implementing a custom class-factory in the referral component. For more details, see the "DCOM Architecture" White Paper.

As user demand grows, administrators can change the components to transparently choose different servers for different users. Client components remain entirely unchanged, and the application can migrate from a model whose administration is decentralized to a centrally administered approach. DCOM's location independence and support for efficient referral make this kind of design flexibility possible.

Dynamic Load Balancing

Static load balancing is a good technique for dealing with growing user demand, but it requires the intervention of an administrator and works well only for predictable loads.

The idea of the referral component can be used to provide more intelligent load balancing. Instead of just basing the choice of server on the user ID, the referral component can use information about server load, network topology between client and available servers, and statistics about past demands of a given user. Every time a client connects to a component, the referral component can assign it to the most appropriate server available at that moment. Again, from the client's point of view this all happens transparently. This method is called dynamic load balancing.

For some applications, dynamic load balancing at connection time may not be sufficient. Clients may not typically disconnect for long periods of time, or demand may be unevenly distributed among users. DCOM does not, by itself, provide support for this kind of dynamic reconnection and distribution of method invocations, since doing so requires intimate knowledge of the interaction between client and component: The component typically retains some client-specific status information (state) between method invocations. If DCOM suddenly reconnected the client to a different component on another machine, this information would be lost.

However, DCOM makes it easy for application designers to introduce this logic explicitly into the protocol between client and component. The client and the component can have special interfaces to decide when a connection can safely be rerouted to another server without loss of any critical state information. At this point, either the client or the component can initiate a reconnection to another component on another machine before the next method invocation. DCOM provides all the rich protocol extensibility mechanisms necessary to implement these additional application-specific protocols.

The DCOM architecture also permits injecting component-specific code into the client process. Whenever the client invokes a method, a proxy component provided by the real component intercepts this invocation in the client process and can reroute it to other servers. The client does not have to be aware of this at all; DCOM provides flexible mechanisms to transparently establish these "distributed components." For details, see the section on custom marshaling in the "DCOM Architecture" White Paper.

With this unique feature, DCOM makes possible the development of generic infrastructures that deal with load balancing and dynamic method routing. Such an infrastructure can define a standard set of interfaces that convey the presence or absence of state information between a client and a component. Whenever the client-side part of the component detects absence of state information, it can dynamically reconnect the client to a different server.

Example: Microsoft's Transaction Server (formerly code-named "Viper") uses this mechanism to extend the DCOM programming model. By requiring a simple set of standardized state information management interfaces, Transaction Server can obtain the necessary information to offer sophisticated load balancing. In this new programming model, client and component interactions are bundled into transactions that basically indicate when a sequence of method invocations has reached a point where no state information remains pending between the two components.

DCOM provides a powerful infrastructure for implementing dynamic load balancing. Simple referral components can be used to transparently implement dynamic server allocations at connection time. More sophisticated mechanisms for rerouting individual method invocations to different servers can easily be implemented, but they require more intimate knowledge of the interaction between clients and components. Microsoft's Transaction Server ("Viper"), built entirely on DCOM, provides a standardized programming model that conveys this additional application-specific knowledge to the Transaction Server infrastructure, which in turn can perform very sophisticated static and dynamic reconfiguration and load balancing.

Fault Tolerance

Graceful failover and fault tolerance are vital for mission-critical applications that require high availability. Such resilience is usually achieved through a number of hardware, operating system, and application software mechanisms.

DCOM provides basic support for fault tolerance at the protocol level. A sophisticated pinging mechanism, described in Section 0, "Shared Connection Management Between Applications," detects network and client-side hardware failures. If the network recovers before the timeout interval, DCOM reestablishes connections automatically.

DCOM makes it easy to implement fault tolerance. One technique is the referral component introduced in the previous section. When clients detect the failure of a component, they reconnect to the same referral component that established the first connection. The referral component has information about which servers are no longer available and automatically provides the client with a new instance of the component running on another machine. Applications will, of course, still have to deal with error recovery at higher levels (consistency, loss of information, etc.).

With DCOM's ability to split a component into a server side and a client side, connecting and reconnecting to components, as well as consistency, can be made completely transparent to the client.

Example: Microsoft's Transaction Server ("Viper") provides a generic mechanism for handling consistency at the application level. Combining multiple method invocations into atomic transactions guarantees consistency and makes it easier for applications to avoid loss of information.

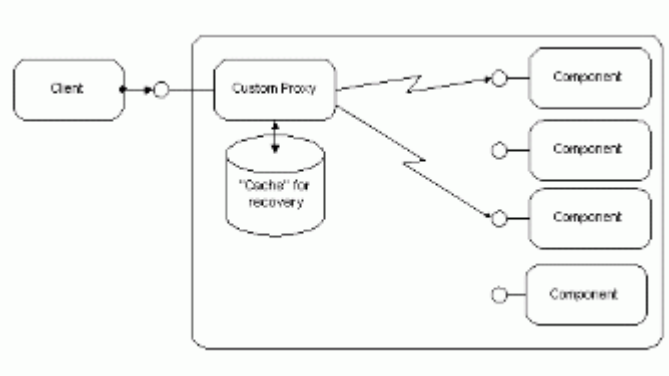

Figure 15: - Distributed component for fault-tolerance

Another technique is commonly referred to as "hot backup." Two copies of the same server component run in parallel on different machines, processing the same information. Clients can explicitly connect to both machines simultaneously. DCOM's "distributed components," make this action completely transparent to the client application by injecting server code on the client-side, which handles the fault-tolerance. Another approach would use a coordinating component running on a separate machine, which issues the client requests to both server components on behalf of the client.

A failover attempts to "migrate" a server component from one machine to the other when errors occur. This approach is used by the first release of Windows NT Clusters, but it can also be implemented at the application level. DCOM's "distributed components" make it easier to implement this functionality and shields clients from the details.

DCOM makes implementing sophisticated fault-tolerance techniques easier. Details of the solution can be hidden from clients using DCOM's "distributed components," which run part of the component in the client process. Developers can enhance their distributed application with fault-tolerance features without changing the client component or even reconfiguring the client machine.

Ease of Deployment

The best application is useless, if it can not be easily installed and administered. For distributed applications, it is critical to be able to centralize administration and make client installation as straightforward as possible. It is also necessary to provide administrators with ways to detect possible failures as soon as possible, preferably before they cause any damage.

How can DCOM help in making an application more manageable?

Installation

A common approach to simplifying client side installation can be summarized under the buzzword "thin client": the less functionality that resides on the client, the fewer installation and maintenance problems can occur.

However, the "thinner" the client components, the less user-friendly the overall application, and the higher the demands to both network and server. Also, thin clients do not take advantage of the significant computing power already available on today's desktops, which is not likely to decrease for most users because desktop productivity applications like word processors or spreadsheets are inherently monolithic. Choosing the right level of "thickness" is thus a critical decision in the design of a distributed application.

DCOM helps in making this tradeoff between flexibility and ease of deployment by letting developers and even administrators choose the location of individual components. The same business components (for example, data entry validation) can be run on the server or on the client with a simple change of configuration. The application can dynamically choose which user interface component to use (HTML generator on the server or ActiveX control on the client).

The biggest problem for maintaining "fat" clients is updating those clients to newer versions. As of today, Microsoft Internet Explorer 3.0 provides an elegant solution to this problem with its support for code downloading. Whenever a user browses to a page, Microsoft Internet Explorer checks the version of the ActiveX controls used on the page and updates them automatically if needed. Applications can also use this support directly (the ActiveX **CoCreateClassFromURL** function) without explicitly using a browser.

In Windows NT 5.0, the concept of code download will be extended to the concept of a native COM class store. This class store will use the extended directory to store configuration information about components, including references to the actual code, conceptually extending the local registry as used today. The class store will effectively provide both Intranet (extended directory) and Internet (code download, Internet search-path) code repositories, making them completely transparent to existing applications.

Installing and updating the server components is usually a much less critical problem. However, in a highly distributed application, it is often not possible to upgrade all clients simultaneously. DCOM's robust versioning support, described in Section 0, "Evolving Functionality: Versioning,") allows servers to expose new functionality while maintaining complete backward compatibility. A single server component can handle both old and new clients. Once all clients are updated, the component can phase out support for the functionality that is not needed by the new clients.

With both code-download and its future extension, the class store, administrators can centrally install and upgrade clients efficiently and robustly, making it possible to migrate from "fat"

clients to intelligent clients without thinning out too much functionality. DCOM's support for robust versioning makes it possible to update servers without previously updating all potential clients.

Administration

Part of installing and upgrading client components is configuring those components and maintaining their configuration. As far as DCOM is concerned, the single most important configuration information is the server machine that runs the components needed by a client.

With code download and the class store, this configuration information can be managed from a central location. A simple change to the configuration information and installation packages updates all the clients transparently.

Another technique to manage client configuration is through the use of the referral components described in Section 0, "Load Balancing." All clients connect to this referral component, which contains all the configuration information and returns the appropriate component to each client. Simply changing the central referral component changes all clients.

Some components, typically server components, require additional component-specific configuration. These components can use DCOM to expose additional interfaces, which allow changes to the configuration and retrieval of the current configuration. Using DCOM's security infrastructure, developers can make these interfaces available only to administrators with the appropriate access permissions. The broad support for rapid development tools makes it easy to write elegant front-end applications that use the administrative interfaces. The same interfaces can be used for automated configuration changes using simple scripting languages like Visual Basic Script or Java Script.

Code download and the class store can be used to centrally configure components. Referral components are an efficient and elegant way to further centralize configuration information. Components can expose additional DCOM interfaces only visible and accessible to administrators, allowing the same DCOM infrastructure to be used for configuration and monitoring of components.

Protocol neutrality

Many distributed applications have to be integrated into a customer's or corporation's existing network infrastructure. Requiring a specific network protocol would require an upgrade of all potential clients, which is simply unacceptable in most situations. Application developers have to take care in keeping the application as independent as possible of the underlying network infrastructure.

DCOM provides this abstraction transparently: DCOM can use any transport protocol, including TCP/IP, UDP, IPX/SPX and NetBIOS. DCOM provides a security framework on all of these protocols, including connectionless and connection-oriented protocols.

Developers can simply use the features provided by DCOM and be assured that their application is completely protocol-neutral.

Platform neutrality

A distributed application often has to integrate different platforms on both the client side and the server side. Developers are confronted with significant differences in many aspects of those platforms: different user interface philosophies, different system services, and even the set of available network protocols make it difficult to target and integrate multiple platforms.

One approach to this problem is to choose the lowest common denominator of all the platforms and use an abstraction layer to maintain a single code base for all platforms. This approach is taken by many conventional cross-platform development frameworks, as well as virtual machine environments like Java. Its appeal lies in having a single code base or even a single binary for all the supported platforms.

However, this simplicity comes at a price. The abstraction layers introduce additional overhead and prevent the use of powerful platform-specific services and optimizations. For user-interface components, this approach often means poor visual integration with other applications, resulting in greater difficulty of use and increased training costs. For server components, this approach sacrifices the ability to tune the performance of critical components for any platform.

DCOM is open to all approaches to cross-platform development. It does not preclude the use of platform-specific services or optimizations, nor does it favor a certain style of system services over others.

DCOM's architecture allows the integration of platform-neutral development frameworks and virtual machine environments (Java), as well as high-performance, platform-optimized custom components into a single distributed application.

Per-Platform Binary Standard

On one hand, DCOM defines a per-platform binary standard, so customers and developers can mix and match components generated with tools from different vendors and use them with different implementations of the DCOM runtime itself. Even though the details of the DCOM run-time library (Object Request Broker) may vary from implementation to implementation, the interaction between the run-time library and the components, as well as between components, is standardized. Unlike other, more abstract object models, with DCOM it is possible to distribute a single binary version of a component for a given platform that works with all other components and run-time libraries.

Cross-Platform Interoperability Standard

On the other hand, DCOM defines cross-platform services (or abstractions) for object-oriented distributed computing, including connection to, and creation of, components, locating components, invoking methods on components, and a security framework.

Beyond this, DCOM simply uses the services available on each platform to implement multithreading and concurrency control, user interface, file system interaction, non-DCOM network interaction, and the actual security provider.

Making the most of DCE RPC

The DCOM wire-protocol is based on DCE RPC, so it is easy to implement DCOM on platforms for which DCE RPC is already available. DCE RPC defines a proven standard for converting in-

memory data structures and parameters into network packets. Its Network Data Representation (NDR) is platform neutral ("reader makes right") and provides a rich set of portable data types.

COM and DCOM also borrow the notion of globally unique identifiers (GUIDs) from DCE RPC. GUIDs provide collision-free, unmanaged naming of objects and interfaces and are the basis of COM's robust versioning.

DCOM's pluggable security providers enable seamless integration with DCE-based security environments. Windows NT 4.0 can serve today as a gateway between platforms supporting ORPC-enhanced DCE RPC (DCOM) and platforms that provide only standard DCE RPC support. This is very useful for integrating existing DCE RPC-based applications on other platforms, and it provides a smooth migration path for multitier applications that can incrementally take advantage of DCOM features.

Available Platforms

DCOM on Windows

Implementations of DCOM are available today on the Microsoft Windows NT platform in Windows NT Workstation 4.0 and Windows NT Server 4.0. DCOM for Windows 95 is currently in beta and due for release before the end of 1996.

DCOM on Apple Macintosh

Microsoft is working on an implementation of DCOM for the Apple Macintosh. A beta version will be available in early 1997.

DCOM on UNIX/Mainframe

Versions of DCOM for UUNIX platforms are under development (Software AG and Digital in close cooperation with Microsoft) and have been previewed publicly interoperating with implementation on Windows NT 4.0.

Beta versions for Sun Solaris, AIX, MVS, and Linux will be available in late 1996. Release versions arescheduled for the first half of 1997.

DCOM and Java

Microsoft provides the reference implementation of the Java Virtual Machine for the Windows platform. This Java VM is included in Microsoft Internet Explorer 3.0 and provides full support for COM and DCOM.

Any Java applet is automatically exposed as a DCOM component and can be used within a Web Page or as a standalone component.

Java applications can also use most DCOM components by simply importing a Java-compliant description of the component, a description that is automatically generated from a type library. See Most of the papers referenced in this whitepaper are available on the Technology Preview CD. URLs to documents which are available for download are listed below.

[DCOM - Architecture] for details. The DCOM component appears to the Java programmer like any other Java class.

As DCOM becomes available on other platforms, the Java VM on these platforms can easily be extended to provide the same level of DCOM integration.

DCOM and CORBA

The Common Object Request Broker Architecture (CORBA) is a competing standard for distributed object computing. It defines an abstract object model that describes components and their interfaces. It also provides standard mappings from the abstract object definition to concrete programming languages, but it does not define a binary standard in any way.

Different ORB implementations that adhere to the standard can achieve, at the most, source level compatibility, but not interchangeability of binary components, one of the reasons that there is not, and probably will not be, a market for off-the-shelf reusable components. Component providers have to provide source code for their components or compile and test not only for each target platform, but for each target ORB implementation.

CORBA also defines a standard for inter-ORB communication that allows two compliant ORB implementations to invoke methods on objects on each other's machine.

ORB implementations like IBM's DSOM and Iona's Orbix commonly provide proprietary extensions to the object model, the language bindings, and the inter-ORB protocol. To take full advantage of a given platform, developers have to sacrifice cross-ORB interoperability and cross-ORB portability.

CORBA also defines a separate inter-ORB communication protocol called IIOP, which is targeted for use on the Internet. It accommodates the intersection of all inter-ORB protocols.

Please refer to [COM and CORBA] for more details.

Seamless Integration with other Internet protocols

At its core, the Internet is a global, decentrally managed and shared TCP/IP network. It makes global connectivity a commodity. The "killer application" that caused the Internet to accumulate critical mass is a simple, standardized page description language (HTML), together with a simple document download protocol (HTTP).

Distributed applications can take advantage of the Internet in many different ways.

DCOM Over Virtual Private Networks

At the lowest level, using the Internet as a cheap, global TCP/IP network opens new opportunities for companies to connect remote sites and individual users.

Virtual private networks such as the Windows NT 4.0 Point-to-Point Tunnelling Protocol (PPTP) are one way of using the network to securely tunnel private information over the Internet. DCOM-based applications can transparently leverage such a virtual private network.

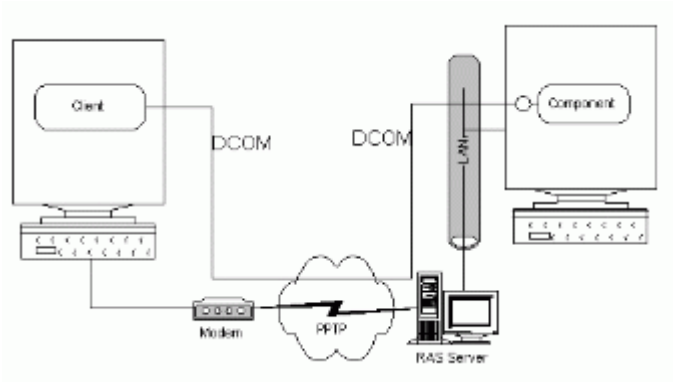

Figure 16: - DCOM over Virtual Private Network

DCOM Over the Internet

Since DCOM is an inherently secure protocol, it can be used without being encapsulated in a virtual private network: DCOM applications can simply use the cheap, global TCP/IP network. Most companies do not provide direct Internet access to their desktop computers. All but some dedicated server machines are hidden behind a firewall that typically consists of protocol-level (port-based) and application-level (proxy servers) filters. DCOM can work well with both classes of firewalls:

- DCOM uses a single port for initiating connections and assigns a configurable range of ports to the actual components running on a machine (actually one port per process. See the "DCOM Architecture" White Paper for details.)
- Application-level proxies can easily be built. They can be either generic (forwarding configurable DCOM activation and method calls) or application-specific.
- Server administrators can also choose to tunnel DCOM through HTTP, effectively bypassing most of today's firewalls.

With this range of options, DCOM-based applications can use the Internet for private connectivity within a company, private communication with external customers and partners, and massive public connectivity to any client in the world. In each of these scenarios, DCOM provide flexible security whenever it is needed.

Integrating HTML and Distributed Computing

Besides just using the Internet as a cheap TCP/IP network, distributed applications can also take advantage of existing standard protocols and formats wherever that makes sense. For noninteractive, textual, or simple graphical information, HTML pages are a great vehicle that provide a well-known and easy way for users to access the information they require.

For more complex, structured, and interactive information, custom components can extend the HTML page to perform the real tasks of the distributed applications in a user-friendly, secure, and efficient manner. Simple business rules can be applied on the client side, providing immediate feedback to the user. More complex business rules can transparently involve server components over DCOM. Thanks to DCOM's language neutrality, these components can be implemented in virtually any programming language, including C++, Java, Visual Basic, or Cobol. Existing off-the-shelf components (ActiveX controls) can be tied to client-side or even server-side custom components using Visual Basic Script or JScript. Please refer to Most of the

papers referenced in this whitepaper are available on the Technology Preview CD. URLs to documents which are available for download are listed below.

[DCOM - Architecture] and [ActiveX] for more details.

Regardless of whether the developer enriches a distributed application with HTML elements or enriches a HTML-based "application" with elements of distributed computing, DCOM provides the necessary component glue to tie the different worlds together.

Summary

DCOM makes it easy to write a distributed application that

- Scales from the smallest single computer environment to the biggest pool of server machines.
- Provides rich, symmetric communication between components.
- Can be robustly expanded to meet new functional requirements.
- Takes advantage of existing custom and off-the-shelf components.
- Integrates teams proficient in any programming language and development tool.
- Uses network bandwidth carefully, while providing great response times for end-users.
- Is inherently secure.
- Provides a smooth migration path to sophisticated load-balancing and fault-tolerance features.
- Can be efficiently deployed and administered.
- Can be used with any network protocol and integrated into any hardware platform.
- Can seamlessly take advantage of other Internet standards and protocols.

DCOM is the TCP/IP of objects.

For More Information

For the latest information on Windows NT Server, check out our World Wide Web site at http://www.microsoft.com/backoffice or the Windows NT Server Forum on the Microsoft Network (GO WORD: MSNTS).

References

Most of the papers referenced in this whitepaper are available on the Technology Preview CD. URLs to documents which are available for download are listed below.

References

Most of the papers referenced in this whitepaper are available on the Technology Preview CD. URLs to documents which are available for download are listed below

[DCOM - Architecture]	
[DCOM - Solutions in Action]	

[COM Spec]	The Component Object Model specification: <http://www.microsoft.com/oledev/olecom/title.htm>
[DCOM RFC]	The DCOM network protocol, submitted as an informational RFC: < http://ds1.internic.net/internet-drafts/draft-brown-dcom-v1-spec-00.txt>
[ActiveX]	The ActiveX Working Group: <http://www.activex.org>. Microsoft's ActiveX Resource Area: <http://www.microsoft.com/activex>
[COM and CORBA]	<http://www.microsoft.com>
[Security]	
[Transaction Server]	