components can communicate with each other across processes in a single computer or between computers over the Internet.

However, components by themselves do not solve all of the issues of enterprise application complexity. For example, suppose a business wants to rapidly build and deploy a customer order entry application that involves five different areas of functionality: tax calculation, customer credit verification, inventory management, warranty update, and order entry. The application will be built from five separate components and will operate on a Web server. How does the developer handle exceptions? System failures? Network outages? Peaks in performance load? Must these be hand-coded into the application? It defeats the two main goals of component-based development—fast time to market and lower development costs—if companies are forced to hand-code the mission-critical services that are required for online production systems.

To address enterprise requirements for a distributed component architecture without sacrificing rapid development and cost effectiveness, the following standardized architectures support this requirement.

Hundreds of applications and thousands of their objects (components) are distributed through the e-enterprise environment. To facilitate this distribution, particularly among objects (components) from software developed by different programmers and vendors, standards have been offered by some developers, such as OMG (CORBA), Sun Microsystems (EJB), Microsoft (COM), and others.

CORBA STANDARD

CORBA, which stands for Common Object Request Broker Architecture, is an industry standard developed by the OMG (Object Management Architecture Guide, a consortium of about 800 companies organized in 1989). CORBA is open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network. Some large companies are embedding CORBA in networked devices for finance and medical applications.

CORBA is useful in many situations. Because of the easy way that CORBA integrates machines from so many vendors, with sizes ranging from

mainframes through minis and desktops to hand-held and embedded systems, it is the middleware of choice for large (and even not-so-large) enterprises. One of its most important, as well most frequent, uses is in servers that must handle large numbers of clients, at high hit rates, with high reliability. CORBA works behind the scenes in the computer rooms of many of the world's largest Websites, ones that you probably use every day. Specializations for scalability and fault-tolerance support these systems. But it is not used just for large applications; specialized versions of CORBA run real-time systems and small embedded systems. In CORBA, client and object may be written in different programming languages.

CORBA's architecture is based on *Object Orientation,* and built around seven key building blocks (www.omg.org):

- OMG Interface Definition Language, OMG IDL – defines the types of objects by defining their interfaces. An interface consists of a set of named operations and the parameters to those operations. Despite the fact that IDL is similar to C++ and Java, IDL is not a programming language. Through IDL, a particular object implementation tells its potential clients what operations are available and how they should be invoked. From IDL definitions, the CORBA objects are mapped into different programming languages, such as C, C++, Java, Smalltalk, LISP, and Python,

- Dynamic Invocation Interface (DII) – it allows client applications to use server objects without knowing the type of those objects at compile time,

- Dynamic Skeleton Interface (DSI) – it is a gateway to a server,

- Interface Repository (IR) – it provides another way to specify the interfaces to objects. Interfaces can be added to the interface repository service. Using the IR, a client should be able to locate an object that is unknown at the compile time, find information about its interface, then build a request to be forwarded through the OBR,

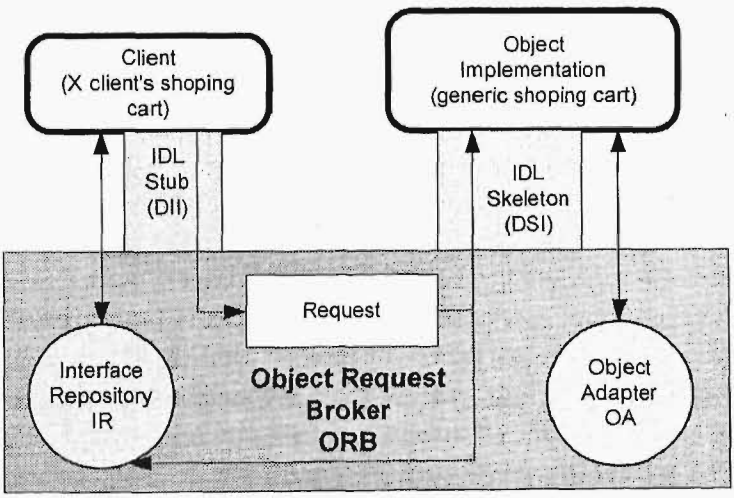
- Object Adapters (OA) – it is the primary way that object implementation access services are provided by the ORB. Such services include: object reference generation and interpretation, invocation method, security of interaction, and object implementation activation and deactivation,

- The Object Request Broker or ORB is a software responsible for: 1) finding the object implementation for the request, 2) preparing the object implementation to receive the request, and 3) communicating the data making up the request. A number of implementations exist in the market today, including ORBIX from IONA Technologies (*www.iona.ie*), VisiBroker from Inprise (*www.inprise.com*), and JavaIDL from JavaSoft (*www.java.sun.com/products/jdk.idl*),

- The standard protocol IIOP (The Internet Inter-ORB Protocol) makes sure that a client will be able to communicate with a server written for a different ORB from a different vendor.

CORBA applications are composed of *objects*, individual units of running software that combine functionality and data, and that frequently (but not always) represent something in the real world. Typically, there are many *instances* of an object of a single *type* - for example, your e-commerce website would have many shopping cart object instances, all identical in functionality but differing in that each is assigned to a different customer, and contains data representing the merchandise that its particular customer has selected. For each object type, such as your shopping cart, you define its interface in OMG IDL.

Figure 5-3: A Request Passing from a Client to an Object's Implementation



This fixes the operations it will perform and the parameters (input and output) for each. This interface definition is independent of your programming language, but *maps* to all of the popular programming languages via a set of OMG standards: OMG has standardized mappings for C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript.

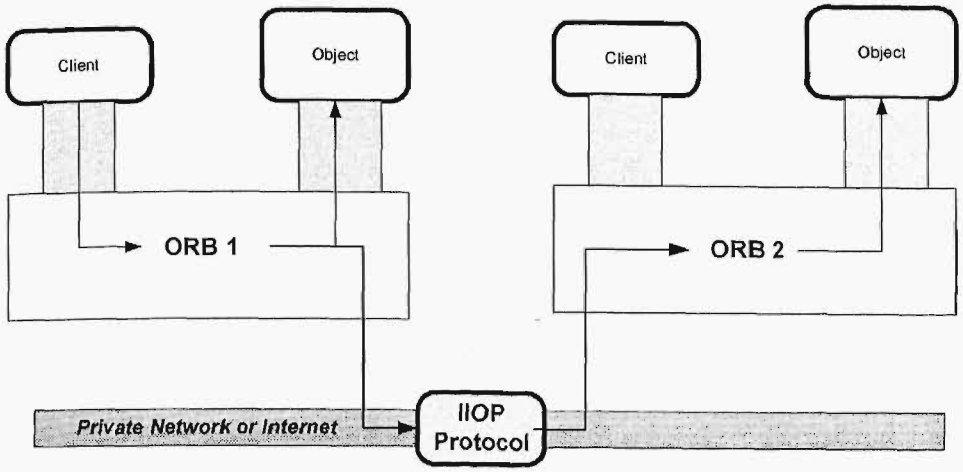
This is the essence of CORBA - how it enables interoperability, with all of the transparencies we have claimed. The *interface* to each object is defined very strictly. But, in contrast, the *implementation* of an object - its running code, and its data - is hidden from the rest of the system (that is, *encapsulated*) behind a boundary that the client may not cross. Clients access objects only through their advertised interface, invoking only those operations which that object chooses to expose, with only those parameters (input and output) that are included in the invocation.

Figure 5-3 shows how everything fits together, at least within a single process: You compile your IDL into client stubs as a Dynamic Invocation Interface (DII) and object Dynamic Skeleton Interface (DSI) and write your object (shown on the right) and a client for it (on the left). DII uses the Interface Repository (IR) to validate and retrieve the signature of the operations on which a request is made. Stubs and skeletons serve as proxies for clients and servers, respectively. Because IDL defines interfaces so strictly, the stub on the client side has no trouble meshing perfectly with the skeleton on the server side, even if the two are compiled into different programming languages, or even running on different ORB's from different vendors.

In CORBA, every object instance has its own unique *object reference*, an identifying electronic token. Clients use the object references to direct their invocations, identifying to the ORB the exact instance they want to invoke (ensuring, for example, that the books you select go into your own shopping cart, and not into your neighbor's). The client acts as if it's invoking an operation on the object instance, but it's actually invoking on the IDL stub which acts as a proxy. Passing through the stub on the client side, the invocation continues through the ORB (Object Request Broker) and the skeleton on the implementation side to get to the object where it is executed.

How do remote invocations work? Figure 5-4 diagrams a remote invocation. In order to invoke the remote object instance, the client first obtains its object reference. (There are many ways to do this, but we will not detail any of them here.) To make the remote invocation, the client uses the same code that it used in the local invocation we just described, but substitutes the object reference for the remote instance. When the ORB examines the object reference and discovers that the target object is remote, it marshals the

Figure 5-4: Inter-operability uses ORB-to-ORB Communication



arguments and routes the invocation out over the network to the remote object's ORB.

Why does this work? OMG has standardized this process at two key levels: first, the client knows the type of object it is invoking (that it is a shopping cart object, for instance), and the client stub and object skeleton are generated from the *same IDL*. This means that the client knows exactly which operations it may invoke, what the input parameters are, and where they have to go in the invocation; when the invocation reaches the target, everything is there and in the right place. We have already seen how OMG has defined this. Second, the client's ORB and object's ORB must agree on a *common protocol* - that is, a representation to specify the target object, operation, and all parameters (input and output) of every type that they may use. OMG has defined this also - it's the standard protocol IIOP. (ORB's may use other protocols besides IIOP, and many do for various reasons. But virtually all speak the standard protocol IIOP for reasons of interoperability because it is required by OMG for compliance.)

Although the ORB can tell from the object reference that the target object is remote, the client cannot. (The user may know this also because of other knowledge - for instance, that all accounting objects run on the mainframe at the main office in Tulsa.) There is nothing in the object reference token that the client holds and uses at invocation time that identifies the location of the target

object. This ensures *location transparency* - the CORBA principle[2] that simplifies the design of distributed object computing applications.

Use of CORBA and UML (Universal Modeling Language) pays off in many types of applications, but here is where the benefits compound:

- If you use CORBA in a small client-server type of application, you will get the benefits of a sound, standard infrastructure, and if you use UML to design before you start to code, you will be much more likely to get a final application with the structure and functionality that you had in mind (or would have asked for!) when you started. CORBA lets you build and run client and server sides on different platforms and in different programming languages, so there are a number of benefits that we can list even for small applications run alone on a network. But these are purely computer-domain benefits; the business functionality of this type of application is the same as if it had been written to sockets, or a proprietary call mechanism.

- Businesses benefit when all of their applications, with their diverse functionality and data, work together. For example, a salesman on the road in a customer's office may need access to product description information (from the catalog), product technical data (from engineering), pricing (from the back office), stock (from the warehouse), production schedule if there are not enough in stock (from the plant), order placement, customer credit data, sales department totals and his own totals, and more. In the old days, he used to collect this by telephone and memo, moving to FAX as technology advanced, but now computer networks give you an opportunity to make the diverse systems in all of these departments work together to support your salesman as he generates the income that, after all, keeps your company in business. So you wrap these legacy applications with OMG IDL interfaces and put them all onto your network, accessible via CORBA object references. This lets your IT department build a client that integrates information from all of them into a sales application. Your salespeople, online in their customers' offices via modem or wireless connection, can answer questions immediately and make more sales. Orders enter into your fulfillment system as they are taken, allowing you to schedule shipping (and production, if you need to) and billing automatically and immediately.

- The sales-peoples' application could become a Web sales site, allowing any potential customer with a browser to find and order your products

themselves. Because all of your supporting applications are available via CORBA, it is easy to generate the application that drives the website, so you do this, and the sales start rolling in.

- With so many customers out there, the load of product support and repair calls increase and, since support has always been a cost center, you brainstorm to figure out a way to deal with it. You finally decide to design a call-center support application around your CORBA enterprise infrastructure. By standardizing response to trouble calls, you deal with these in less time, but the real benefit comes in parts and repair: with all of the engineering diagrams online, your telephone staff can sell and ship parts to customers with one or two clicks of a mouse, and with another click or two, can dispatch contracted repair vans to take care of the installation. Income from parts and commission on repair calls turn this cost center into a profit center, and customers' satisfaction increases at the same time. Integration of many applications around your enterprise that connect to the call-center application, a potential nightmare, is straightforward because they all have CORBA wrappers already.

- With success, the load on your e-commerce website increases until it outgrows the capacity of the mainframe applications that supported it at start-up. Since CORBA has been so successful for you on your network, you decide to build your server's replacement object-oriented in CORBA from the ground up. To start, you use UML for your analysis and design: it helps you gather requirements, work through use-cases, and set down the functionality that the new server will provide. Then, class diagrams, object diagrams, and action diagrams let you picture how it will work. By the time you're done, the UML diagrams that you have generated dictate most of your OMG IDL and language code. To take care of both current and anticipated load, you decide to buy an ORB that runs load-balanced on a roomful of server machines. It is based on OMG's Portable Object Adapter or POA, which helps you take the best advantage of your hardware when you run heavy loads, like the ones that Web-based applications generate. If your application needs to stay up reliably (as it would if you are running stock or bond trades, for example), you may decide to use CORBA Fault Tolerance also. More than just load-balancing, fault tolerance runs every object on two or more separate machines at the same time and automatically switches to the good one if one fails. If you duplicate your hardware (computers, networks, even

power sources) also, you can set up a very reliable server indeed. By the way, you have not replaced all of your legacy systems (that is, the functional systems that run your business!), and this server still needs to access the ones that are still around. Since they all retain their CORBA wrappers, of course, this is routine. To maximize sales, you design the front-end for this server to be accessible from as many customer client types as possible: some customers come in via Web browsers, using HTTP which your Web server translates into CGI invocations of your CORBA front end. Others use Java/CORBA clients for more sophisticated programmatic access, while some users with direct network connections use OMG's standard COM/CORBA bridge to come in straight from a Microsoft desktop. Type-specific adapters condense your screens, eliminating graphics and isolating key lines of text, enabling digital PDA's, pagers, and browser-capable cell phones to place orders using your same business-logic architecture and implementation.

- To keep up with demand, you decide to automate your plant. Because of the speed that parts travel around the assembly line, the control program must run in real-time, so you build this system on an ORB and operating system that conform to the CORBA real-time specification. Computers on your shop-floor equipment also run CORBA (in real-time where needed here too), so your plant hums smoothly along; in fact, some of the processors on your shop floor run minimal CORBA, OMG's standard for embedded ORB's. Of course the plant doesn't run in isolation: it needs stock, which needs to be bought, which costs money, and needs to be brought to the plant, which needs logistical coordination, so your automation system also talks to your supply and logistics applications over CORBA interfaces, of course.

- The next addition would be CORBA-based interfaces to your suppliers, automating your logistics out to their warehouse (and, within their enterprise, hopefully to their plant as well). This is not a purely hypothetical example; Boeing Aircraft automates its manufacturing system from order entry through production and delivery, to maintenance, using CORBA.

So where's the payoff? Just as the different departments and divisions that make up your enterprise need to work together in order to maximize profit, their computing applications also need to work together. CORBA lets you do this, and that is where standards pay off in a big way: you cannot limit the diversity

of computers in your enterprise - you need to pick different computers for different uses. This is clear on the shop floor, where equipment may come with a processor built-in, or in the call center, where machines need to link up to the telephone system computer for both voice and data to take advantage of ID information that comes with incoming calls. So, you pick a *standard interoperability infrastructure* - CORBA! - to tie your applications together. You would like to use it for *all* of your applications, but sometimes this just is not possible. For these cases, CORBA gives you the couplings you need: the COM/CORBA interoperability standard to connect to the desktop that everyone uses (and to Microsoft servers, too, if you have them); the reverse Java-to-IDL language mapping and RMI/IIOP to connect to Java RMI objects and EJB's, and the mapping from XML to OMG IDL.

How do they work together? You'll want (and need, actually) to perform an analysis and design before starting any substantial software development project. For this, you buy and use a tool based on UML, the Unified Modeling Language. Using XML Metadata Interchange or XMI, you transfer your model - which is the *metadata* for your application - into a standardized repository based on the MOF, or Meta-Object Facility. Using XMI again, you transfer your model from your MOF into a development environment that lets you implement it as a CORBA application. You will generate OMG IDL interfaces, which will map into the programming languages that you choose for your clients and objects, using the OMG-standard language mappings. You may design a scalable server-side architecture using features of the Portable Object Adapter or POA, and augment CORBA's support for load-balancing with a standard Fault Tolerant infrastructure. You will surely want to design your overall application around the CORBA services and possibly the Domain CORBA facilities to reap the major gains possible from the buy-vs.-build philosophy.

Who is using them, and what for? OMG has collected hundreds of design wins and success stories from companies that use CORBA, and posted them on our Website at *http://www.corba.org*. For readers who do not want to surf to that page now and get totally sidetracked, here are summaries of a few big CORBA users' applications:

- CNN uses an application based on the CORBA event service to distribute news material that comes in from hundreds of sources, in many formats, from many different machine types, to all of their news editors who run automated filter programs that audit the incoming events and flag the stories that qualify as important to each editor's individual preferences.

- Boeing integrated four "best of breed" manufacturing applications into a comprehensive infrastructure that takes care of airplane configuration from ordering, through manufacture, to maintenance.

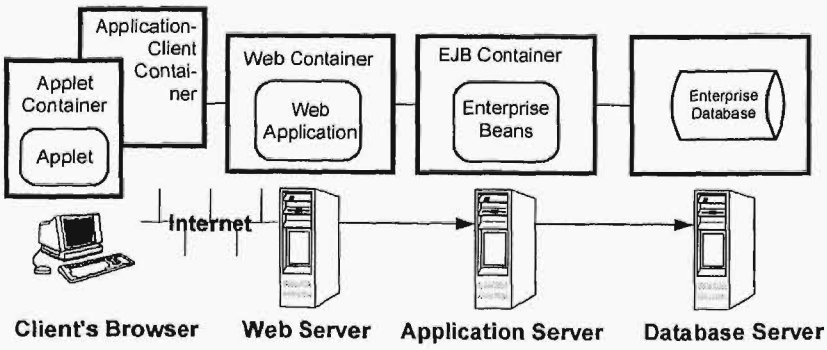
- Charles Schwab and Company built a CORBA-based trading application that they use for their 5,000 best customers, handling accounts worth multiple billions of dollars.

ENTERPRISE JAVABEANS (EJB) STANDARD

The application of the Internet and its Web technology has changed the traditional client-server two-tier architecture into the four-tier architecture. Sun Microsystems, which developed the Java language, supports this architecture on its servers. Web-driven applications use various plug-in extensions to Web servers. These extensions invoke programs on the server that dynamically construct HTML documents ("home pages") from information stored in corporate databases and vice versa, the Web server extensions also enter information submitted in HTML forms into the corporate databases. An example of such extensions is CGI-bin scripts, which stands for Common Gateway Interface, and interface for developing HTML pages and Web applications.

Figure 5-5 illustrates the so-called J2EE (Java 2 Enterprise Edition) architecture which allows for the development and implementation of enterprise Web-oriented applications using the Java programming language.

Figure 5-5: The Four-Tier J2EE Architecture for Web-based Applications



The J2EE platforms consist of four programming environments, called containers:

- The EJB Container – provides the environment for the development, deployment, and runtime management of enterprise beans. Enterprise beans are components that implement the business processes and entities under the form of applications.

- The Web Container – provides the environment for the development, deployment, and runtime management of servlets and JavaServer Pages. Servlets are specialized programs called pseudo applets (mini applications) that run on the server side. Java Servlets are a popular choice for building interactive Web applications, replacing the use of CGI scripts. Servlets are similar to applets (generated from the browser) in that they are runtime extensions of applications. Instead of working in browsers (like applets), servlets run with Java Web servers, configuring or tailoring the server.

- The Application-Client Container – provides the environment for executing J2EE application clients. This environment is essentially the Java 2 platform, Standard Edition.

- The Applet Container – provides the environment for executing Java applets. This environment is typically embedded in a Web browser.

The J2EE Platform embraces the Common Object Request Broker Architecture (CORBA). All J2EE Containers include a COBRA-compliant Object Request Broker (ORB) module. The inter-operability protocol between EJB Containers from multiple vendors is based on COBRA standards, such as Remote Method Interaction over the Internet Inter-ORB protocol (RMI-IIOP) and the Object Transaction Service (OTS).

The EJB architecture requires six roles of professionals:

1. Bean Developer – develops the enterprise applications' components (beans),

2. Application Assembler – puts together different, but logically interrelated beans into a larger unit such as a subsystem or a system,

3. Deployer – deploys the application within a particular computer operated environment,

4. System Administrator – configurates and administers the Enterprise Information Infrastructure,

5. EJB Container Provider and EJB Server Provider – a vendor specializing in transactions and application management.

DNA - DCOM STANDARD[3]

The Distributed Component Object Model (DCOM) developed by Microsoft has three strengths that make it a key technology (*www.msdn.microsoft/library.com*):

- DCOM is based on the most widely-used component technology today. DCOM is simply "COM[4] with a longer wire"—a low-level extension of the Component Object Model, the core object technology within Microsoft® ActiveX® (COM enabled for the Internet). Major development tools vendors—including Microsoft, Borland, Powersoft/Sybase, Symantec, ORACLE, IBM, and Micro Focus—already sell software development tools that produce ActiveX components. These tools and the applications they produce automatically support DCOM, providing the broadest possible industry support. Additionally, over 1,000 existing commercial software components that will work with DCOM are already available for use by developers.

- DCOM extends component applications across the Internet. Because it is an ActiveX technology, DCOM works natively with Internet technologies like TCP/IP, Java, and HTTP, enabling business applications to work across the Web. DCOM enables distributed Java today without requiring any communications-specific code or add-ons.

- DCOM is an open technology that runs on multiple platforms. Microsoft is openly licensing DCOM technology to other software companies to run on all of the major operating systems, including multiple implementations of UNIX-based systems. Software AG has DCOM running on the Solaris-based operating system today. Additionally, Microsoft is handing

over DCOM technology with other core ActiveX technologies to The Open Group. The Internet Draft technical publication that contains a publicly available description of the DCOM network protocol can be found at *http://www.dc.luth.se/doc/id/draft-brown-dcom-v1-spec-00.txt*.

The combination of these three factors—the largest installed base, native support for Internet protocols, and open support for multiple platforms—means that businesses can gain the benefits of a modern component application architecture without having to replace investments in existing systems, staff, or infrastructure.

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.

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 5-6 illustrates how the COM/DCOM run-time libraries provide the link between client and component.

Figure 5-6: DCOM: COM Components in Different Processes (DCE–Distributed Computing Environment, RPC–Remote Procedure Call, LPC–Local Procedure Call)

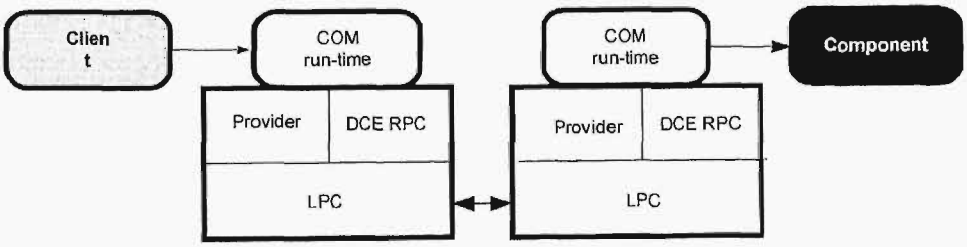
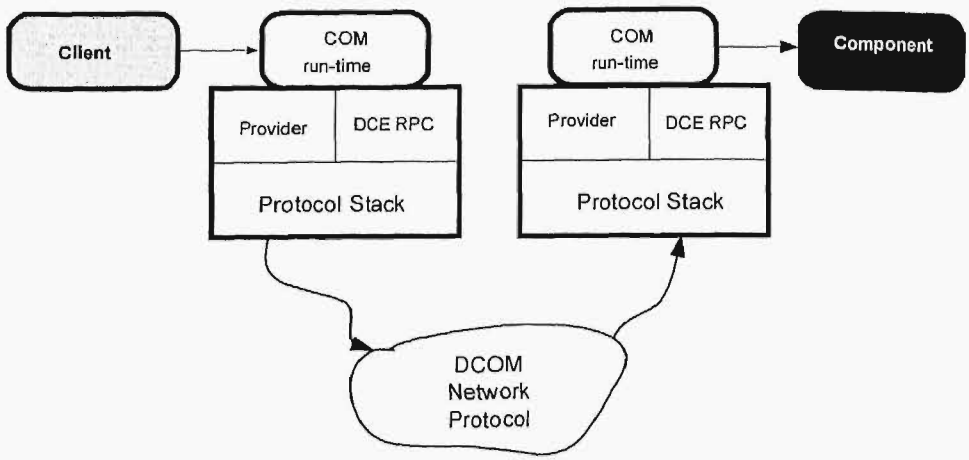


Figure 5-7: DCOM: COM Components on Different Machines



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 is aware that the wire that connects them has just become a little longer. Figure 5-7 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.

Components and Reuse

Most distributed applications are not developed from scratch and in a vacuum. Existing hardware infrastructure, existing software, and 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.

Components and the Enterprise

As distributed applications are built from simple components and Internet protocols emerged, a new set of enterprise platform services for component applications will be required. To address enterprise requirements for distributed component architecture without sacrificing rapid development and cost effectiveness, Microsoft is integrating DCOM into the Active Server. The Active Server is a series of technology services that speed deployment of component-based applications for the Internet and corporate intranets. These services include:

- **Transactions**—traditional rollback and recovery for component-based applications in the event of system failure.
- **Queuing**—integration of component communication with reliable store-and-forward queues, which enables component applications to operate on networks that are occasionally unavailable.
- **Server scripting**—easy integration of component applications on the server with HTML-based Internet applications.
- **Legacy access**—integration of component applications with legacy production systems, including mainframe systems running CICS and IMS.

The Active Server technologies use publicly obtainable Internet protocols and are currently available[5].

MICROSOFT .NET FRAMEWORK[6]

Microsoft evolves from the COM-DNA platform to the new .NET platform designed to simplify application development in the highly distributed environment of the Internet. This is a transformation from desktop applications to the distributed GUI-based .NET applications. The .NET Framework affects all of Microsoft's products, from operating systems, servers, and middleware to applications. All these products are capable of handling and processing .NET traffic and leveraging the .NET infrastructure. These products will transform to Windows.NET, Office.NET, MSN.NET, and so forth.