Distributed Architecture Technologies

Robert Peacock

Moving from a legacy architecture to one that uses distributed objects has both complexities and rewards. IT managers who are planning this move face several distribution issues that require them to make crucial choices involving a wide range of languages, operating systems, networking protocols, and applications.

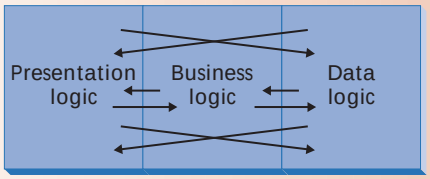
Understanding this process calls for a review of the various types of architectures.

ONE-TIER ARCHITECTURES

A one-tier or monolithic architecture is a traditional mainframe environment. As Figure 1 shows, in this type of legacy architecture, one physical machine encompasses all three fundamental business application areas: presentation logic, business logic, and data logic.

This closed approach offers companies contrasting benefits and disadvantages. The benefits include security, management, and control through centralization. In addition, the architecture can handle a large number of users without jeopardizing performance.

Figure 1. One-tier architecture.



In this closed approach, one machine encompasses all three fundamental business application areas.

Moving to a distributed objects architecture, requires IT managers to make some crucial choices.

One disadvantage is that a one-tier architecture restricts companies to using a single vendor-specific processor. In addition, scaling is costly because the lack of distinction between fundamental applications creates cross-dependencies. The only way to resolve this cross-dependency is to separate the interface from the implementation.

CLIENT-SERVER MODEL

Attempts to resolve the cross-dependency issues inherent in the monolithic architecture evoked a host of technological developments. PCs, LANs, relational databases, desktop tools, and applications all contributed to the development of the two-tier architecture.

Figure 2 shows how the client-server model separates the skill sets into two areas: interface and implementation. This approach essentially divides the existing monolithic architecture by placing the server in a physical location separate from the client. The presentation logic or graphical user interface (GUI) is on the client side, and the implementation or database logic is on the server side. Because this two-tier approach leaves no distinct area where the business logic can reside, this model divides it equally between the client and server. Windows-based PCs and Unix or NT servers often are based on the client-server model.

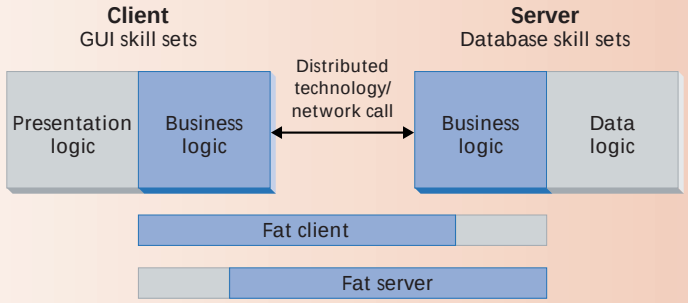
The improved applications and tools in the client-server model facilitate faster development and deployment for the end user compared with a one-tier architecture. Another advantage over one-tier architectures is that Unix servers are smaller and less expensive than larger mainframes.

Implementing a two-tier architecture is simple and it presents no real distribution issues. Dividing the business logic equally between the client and the server facilitates communication between the presentation and business logic on the client side and between the business logic and data logic on the server side.

The two separate business logic locations communicate by using technologies such as MQSeries or Tuxedo to make

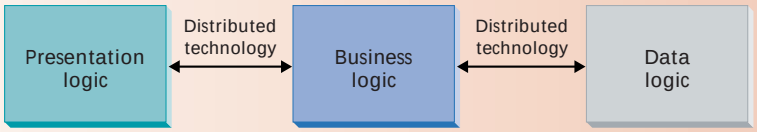
1520-9202/00/$10.00 © 2000 IEEE

Figure 2. Two-tier architecture.



The two-tier client-server model separates the interface (GUI skill sets) and implementation (database skill sets) components into two distinct areas.

Figure 3. Three-tier architecture.



This model makes a clear distinction between the presentation logic, business logic, and data logic components.

networking calls. Another option is to use distributed object technologies to communicate.

Having centralized assets provides a high level of security. However, legacy architectures use available computing resources poorly because they require simple presentation logic and processing to transit the network.

THREE-TIER ARCHITECTURES

A three-tier architecture supports and improves upon the one-tier and two-tier models. This type of architecture overcomes the inconsistencies associated with the distributed business logic location in the two-tier client-server model. It also resolves inconsistencies such as overloading, which causes performance problems.

As Figure 3 shows, the three-tier architecture clearly progresses to the next stage by making a distinction between the presentation logic, business logic, and data logic components. PCs contain the presentation logic and perform simple validation, the business logic executes on the business servers, and centralized data stores and legacy functionality reside on traditional servers. The three separate logic areas use abstract interfaces to communicate. The abstract interface is either a distributed technology (DT) or a distributed object technology (DOT). DOT has advantages over DT because its object principles extend further than a single application's life cycle.

An object is essentially a structure that uses a standardized set of operations and methods to manipulate data. The object-oriented concepts of encapsulation and abstraction create objects that communicate with each other through standardized abstract interfaces. These interfaces identify the operation to be performed and define the input and output parameters required to perform it.

Objects use the abstract interfaces to amalgamate into an integrated application system. Abstract interfaces hide the application logic functionality within the application object. In essence, each object is a separate black box that you can adjust or replace without interfering with the other objects that communicate with it. Any changes to the input and output agents require adjustments to the other components.

The benefits of using objects include versatility, ease of maintenance, and reusability.

COMPONENT TECHNOLOGIES AND OBJECTS

Component-based development and object-oriented development have a number of similarities. On a descriptive level, a component is a physical entity. A component has inherent behavior and attributes, and it answers specific questions: What do I know? What can I do? What is my purpose? Objects also have these traits. One distinction is that a component is a code unit—for example, an .exe or .dll (dynamic link library) file. In contrast, we develop objects individually in a logical fashion, even though we may package them together to produce components.

Developers use traditional languages such as C, assembler, Fortran, and Cobol to write components; they use OO languages such as C++, Java, or Smalltalk to create objects or applications. As a general rule, components have more methods than objects because they are a collection of objects, even though they don't need to export (make visible) all of their methods. By definition, a component is part of something else; unlike an application, it isn't freestanding or capable of its own functionality. However, components are pluggable, so they can provide a sort of standalone functionality.

Components solve low-level design needs, then programmers analyze the components to define additional requirements. Conversely, objects

solve high-level design needs. Objects require OO technologies, a life-cycle approach, and testing. Objects support use cases that are integral to the life-cycle approach.

DISTRIBUTED OBJECT ARCHITECTURE

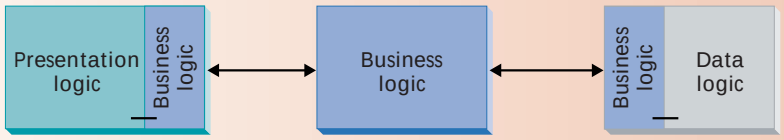
A three-tier distributed object architecture shown in Figure 4 uses middleware to communicate between program objects. The object-based middleware can combine the various enabling technologies to mediate any platform-specific and network translation issues so that programs developed by different vendors can communicate in a network. The "Object-Based Middleware Technologies" sidebar lists various types of middleware specifications that DOT architectures use.

A three-tier architecture has many benefits compared with one-tier and two-tier architectures. The distributed object technology offers the benefits of increased security, reliability, scalability, and availability. Object reuse and sharing substantially increase developer productivity because they minimize the time required for recoding. Abstract interfaces provide more flexibility. With DOT, you can use data and networks more effectively, and the systems are easier to maintain.

A distributed object architecture facilitates communication between different programs in a network. But implementing systems on distributed object middleware can be complex. While middleware provides access, developers still need to implement application services. And on top of today's demands that organizations operate in the e-business arena, IT managers still face the challenge of Web-enabling these applications. ■

Robert Peacock *is director of e-business services at Semaphore, a Massachusetts-based e-business systems integrator. Contact him at rpeacock@sema4usa.com.*

Figure 4. Three-tier distributed object architecture.



Middleware objects mediate platform-specific and network translation issues so that programs developed by different vendors can communicate in a network.

Object-Based Middleware Technologies

Three-tier distributed object architectures use these object-based middleware specifications to mediate platform-specific and network translation issues. In this way, programs developed by different vendors can communicate in a network.

- ***CORBA:*** **The Object Management Group defines and manages the Common Object Request Broker Architecture middleware specification. Founded in April 1989 by 11 companies, OMG has grown to become a consortium of more than 800 organizations. CORBA, a core part of the Object Management Architecture, provides a method invocation mechanism with location and implementation transparency. The OMG's Object Request Broker Task Force has approved the CORBA Component Model specification, with final adoption expected in November 2000.**
- ***COM:*** **The Component Object Model is a set of technologies for building reusable components. Applications use the COM object-based programming model as a framework they can build upon and extend. Designed to promote software interoperability, COM makes the ActiveX object-oriented program technologies and tools possible.**
- ***DCOM:*** **Distributed COM is COM extended over the network. Clients use DCOM to interact with components on other hosts. DCOM also extends COM's location transparency to remote processes. This means that clients don't need recoding to access objects on other systems—"it just works." Further, DCOM supports multiple security mechanisms and transport protocols.**
- ***Java RMI:*** **Java remote method invocation is built into the Java framework. This framework can distribute any Java object that implements the java.rmi.remote interface.**
- ***ODBC/JDBC:*** **Open database connectivity/Java database connectivity provides an object-oriented encapsulation of basic structured query language (SQL) functionality. Just as CORBA, DCOM, and RMI hide the details of networks and transport protocols, so ODBC/JDBC hides the details of a particular database management system.**