Journal of International Information Management

Volume 12 | Issue 1 Article 8

2003

Seleclting Middleware for N-tier applications

Thomas Sandman
California State University, Sacramento

Timothy Riley
Placer County Office of Education

Selecting Middleware for N-tier applications

Thomas Sandman
California State University, Sacramento

Timothy Riley
Placer County Office of Education

ABSTRACT

This paper describes middleware for n-tier architecture, describes how this middleware is meeting the unique demands of Internet applications and e-commerce, and suggests selection guidelines to assist business managers in choosing appropriate types of middleware for n-tier systems that will meet their internet needs. Different types of middleware provide the functionality for addressing many distinct and disparate problems arising from the distributed processing associated with n-tier systems. This paper associates the type of middleware with the nature of the system being developed.

INTRODUCTION

The Internet has grown at a pace so rapid that technology has struggled to keep up. A communications medium that not long ago was read-only, text based-advertising has blossomed into full-scale client-server transaction processing. The conventional technologies that make client-server computing possible are limited in meeting the needs of the Internet. The purpose of this paper is to describe the enabling technologies of n-tier architecture, describe how middleware is meeting the unique demands of Internet applications and e-commerce, and suggest selection guidelines that can assist business managers in choosing the correct mix of building blocks for n-tier systems that meet their internet needs.

MIDDLEWARE - THE ENABLING TECHNOLOGY

Middleware is the enabling technology that has allowed web architects to develop three-tier and n-tier systems, overcoming many problems and limitations inherent in two-tier systems. Many definitions of middleware have been published. It has been defined as, "a vague term that covers all the distributed software needed to support interactions between clients and servers" (Edwards, Harkey, and Orfali, 1999, p. 44), and, as "a layer of software that enables communications between software components regardless of the programming language in which the

components are developed, the protocols used to communicate between components, or the platforms on which the components execute" (Sharon, 1997, p. 92). Bernstein (1996) defines middleware as "a general-purpose service that sits between platforms and applications" (p. 89).

These definitions are quite vague and do not come close to fully describing middleware; thus this section will attempt to paint a clearer picture of what middleware is by explaining the various types of functionality that are offered by middleware applications. Middleware will also be categorized, and an overview of each middleware type, its functionality, advantages and disadvantages will be provided.

Middleware Functionality and Terminology

Middleware can perform many functions in computer information systems. The most common functions performed by middleware are: advanced messaging, application framework capabilities, dynamic load balancing, encapsulation/modularization, legacy integration, multi-database support, naming services, security services, thread pooling, transaction control, transaction queuing, and failover.

- Messaging - Messaging services are provided by all middleware applications. By definition, middleware acts as a layer that allows clients to communicate and transact with servers. However, there are many types of advanced messaging services that can be provided middleware, including the following:
    - Conversational messaging. This allows two applications to pass messages back and forth in a real-time manner (Edwards, 1999). Applications can communicate across the Internet, a WAN, or a LAN with specialized software. TCP/IP Sockets and IBM's CPI-C are examples of the middleware used to support such conversational messaging services.
    - Message queuing. This type of messaging de-couples the client and server interactions by queuing messages on the server (Sharon, 1997) and works well for integrating existing applications. However, there is no transaction control using this method. Therefore, applications that must pass messages in a transactional fashion cannot use message queuing to do so.
    - Publish/Subscribe. This typically allows client applications to subscribe to specific services on a server (Boucher, 1999). The server application then sends messages to an event manager, which determines who or what has subscribed to receive that piece of information and then sends out the information only to appropriate recipients. The publish/subscribe-messaging pattern is becoming widely used for informational internet applications, such as stock or news tickers.
    - Request/reply messaging. This is synchronous, in that an application sends a request for information, and then awaits a reply (Sharon, 1997). Examples of request/reply messaging include remote procedure calls, and object request broker (ORB) remote method invocations, which is part of the CORBA standard. Request/reply works well when simple transaction control is a necessity.

- Application framework capability - Application frameworks are software packages that are designed to assemble applications (typically web applications) quickly using a pre-made set of objects (McFall, 1998). The applications created by these frameworks usually reside on one of the middle tiers located between the client and the server in a web-based multi-tier application. Some application frameworks contain tools used to build business logic for a web application, while others provide a way to assemble quickly a sophisticated graphical user interface for full-scale e-commerce sites.
- Dynamic load balancing - This is the ability of an application to adjust itself based upon the load the server is currently experiencing (Hoffman, 1999). This feature typically allows a middleware application dynamically to create new logical server instances to which information requests can be automatically re-directed. This functionality smoothes out the performance of an application during peak demand periods.
- Encapsulation/modularization - Encapsulation is the ability to assemble program code in a modular, reusable fashion. For purposes of middleware, business logic can be encapsulated into services with simple programming interfaces, thus allowing many applications to use consistent logic when retrieving and performing update transactions on a data store. Security logic can be encapsulated. Objects can be created and interchanged to allow one application to communicate with many different types of data sources. Encapsulation and code re-use are two of the most important functions offered by middleware applications (Bielski, 1999).
- Legacy integration - Middleware also provides the capability to integrate legacy applications into new web applications (Lewis, 1998). By doing so, companies can leverage existing technology to create their web presence. By leveraging existing information systems, companies can realize huge cost savings, because it is not necessary to create a new internet information system from scratch and maintain both systems.
- Multi-database support - Quality middleware applications also provide multi-database support that allows one application to perform transactions across multiple, heterogeneous data sources (Dogac, Dengi, and Oszu, 1998). Full transaction control can be performed across these systems. Web-based multi-tier systems can be designed to integrate marketing, sales, inventory, accounting, payroll, and other corporate systems, which were previously independent of one another.
- Naming services - Naming services provides a way for a client application to connect to an object or web site without having to know the physical location (Garber, 1999). This provides "location transparency" for server applications and objects. Clients do not need to know where a server component is. The naming service stores this information and directs the client to the appropriate server/location automatically.
- Security services - These services provide user authentication capabilities and message encryption, which are two of the most important features for e-commerce sites doing business on the Internet today (Eckerson, 1995). Without proper security, performing any type of financial transaction on the Internet would expose both the client and server to the possibility of electronic theft.
- Thread pooling - Thread pooling allows multiple users to share a single connection to a

data store, thus improving performance and preventing server overload (Edwards, 1999). Most middleware packages that offer this feature can automatically determine how many users should share a connection, and many also allow system administrators to configure the number of users that share a connection to a data store, thus giving them the ability to customize their system based on the capabilities of their system.

- Transaction control - Transaction control is the capability of an application to perform a cohesive update to one or more data sources (Edwards, 1999). What this means is that a transaction is ensured of completion from beginning to end. If any part of the transaction fails, the entire transaction is "rolled back." By rolling back a transaction, the data is restored to its original state, thus preventing data corruption. If all components of the transaction are successfully completed, then the transaction is "committed." When a transaction is committed, changes to the data are applied to the data store. After a transaction is committed, it cannot be rolled back.
- Transaction queuing - Transaction queuing is the ability of an application to serialize or queue transactions. This allows many requests to be submitted to a data source at once, without locking up or overloading the server. Transaction requests are held in queue in a first-in, first-out (FIFO) order of precedence until the middleware application allows them to be submitted (Edwards, 1999). Many applications allow the order of precedence to be overridden based on different levels of priority that can be assigned to a transaction.
- Failover - Another functionality provided by many middleware applications is the failover capability. Failover is the ability to redirect network traffic to a backup server when a primary server goes down or is unavailable (Edwards, 1999). Many applications also include automatic recovery and redirection back to a primary server when it becomes available again.

It is important to note that the above list is not all-inclusive, however, this list is representative of many common features sought after by companies shopping for middleware. It should also be said that during the course of researching this paper, no middleware packages were found to be a comprehensive solution that incorporates all of the functionalities listed above. The next section will discuss the major categories of middleware, and it will indicate which types of functionality are typically found in each category.

Middleware Classification

The features presented above are packaged in different types of middleware packages are available. The primary types of middleware on the market are: database access middleware, RPC (remote procedure call) middleware, DTPM (Distributed Transaction Processing Monitor) middleware, message-oriented middleware (MOM), object request brokers (ORBs), object monitors, application servers, and enterprise application integration middleware (EAI). This section will discuss each type of middleware and the types of functions each performs.

- Database Access Middleware - Database access middleware allows client applications to access data and perform transactions on various types of databases (International

Systems Group, 1997). Database access middleware is commonly used in two-tier applications. An example of database access middleware is ODBC.

- RPC Middleware - RPC middleware is one of the earliest forms of middleware to become available. RPC middleware is based on a mechanism that has evolved from UNIX operating systems (International Systems Group, 1997). These types of middleware have been used for many years to create distributed computing systems, and RPC middleware is considered an industry standard type of middleware, and will be discussed in the next section.
- DTPM Middleware - This is a type of middleware that originated from mainframe computing. DTPM has evolved from transaction processing (TP) monitors, a type of software used on mainframes to ensure transactional integrity against data stores (Edwards, 1999). DTPMs play the same role for client/server systems, ensuring that transactions performed on distributed databases are performed in such a way that the ACID properties, and therefore transactional integrity, are guaranteed (see Table 1). Distributed transaction processing middleware is a mature segment of the market that has evolved in accordance with industry needs; therefore, DTPMs have many advantages. By definition, DTPMs provide complete transaction control. In addition, many DTPMs provide load balancing, failover, location transparency, thread pooling, concurrency control, transaction queuing, and auto restart capabilities (Edwards, 1999). DTPMs have a few disadvantages. Most packages focus on database transactions, and few, if any, provide advanced messaging services, such as request/reply, conversational messaging, publish/subscribe, or message queuing (International Systems Group, 1997). Also, most DTPMs do not provide support for objects.

Table 1. ACID Properties (after International Systems Group, 1997, p. 10)

Atomicity	All operations that an application performs which involve updates to any kind of resource are grouped into a so-called "unit of work." This unit is referred to as atomic, meaning it is indivisible. In other words, the entire unit of work is either performed or not. Any partial completion (due to system failures) will be rolled back.
Consistency	At the end of a transaction, all resources that have participated will be in a consistent state.
Isolation	Concurrent access to shared resources by different units of work (performed by different applications) is coordinated so that they do not affect each other. In other words, transactions that compete for resources are isolated from each other.
Durability	All updates to resources that have been performed within the scope of a transaction will be persistent, or durable.

- MOM Middleware - MOM represents a wide variety of middleware products used to facilitate the transfer of messages between software applications (International Systems Group, 1997). This type of middleware typically provides either synchronous (blocking) or asynchronous (non-blocking) communication, or in some cases both (International Systems Group, 1997). Most MOMs also provide some form of advanced messaging. The software will fall into three general categories: message passing middleware, message queuing middleware, and publish/subscribe middleware.
    - Message passing middleware facilitates a direct connection between two or more programs (International Systems Group, 1997). This connection generally must remain persistent, and therefore this type of middleware does not work well for communication between loosely coupled programs. Most message passing middleware packages support both synchronous and asynchronous messaging.
    - Message queuing middleware uses message queues to facilitate the transfer of messages from one program to another. This eliminates the tight coupling between the communicating programs, and persistent connection is generally not required. This model is frequently used between software applications that do not have a quality connection. Generally, only asynchronous messaging is supported. Some message queuing middleware packages offer thread pooling and persistent queuing. Persistent queuing refers to the ability to recover and resume message queue operation after a system failure (International Systems Group, 1997).
    - Publish/subscribe middleware allows client applications, or subscribers, to subscribe to information from a server, or publisher. Once the subscriber registers with the publisher, the subscriber does not have to poll the publisher for new information, and new information is automatically sent to the subscribers when necessary (International Systems Group, 1997).

  The benefits of advanced messaging middleware are straightforward. Based on the type of messaging middleware selected, users can implement message/passing, message queuing, or publish/subscribe messaging on their systems. In addition, some integrated packages are available which allow all types of advanced messaging. Disadvantages of this type of middleware are that it is very limited, many packages only address messaging, and transaction control is not addressed. Load balancing, failover, location transparency, concurrency control, transaction queuing, and auto restart are generally not addressed.
- Object Request Brokers - Object oriented middleware, commonly referred to as object request brokers (ORBs) represents a type of middleware that supports the transfer of encapsulated components, or objects, between software applications. Object request brokers generally fall into two categories: CORBA-compliant ORBs, and Microsoft COM objects (Gaudin, 1997). Pure object request brokers simply provide a standard means by which encapsulated objects communicate with one another. Object request brokers typically do not provide transaction processing services, security services, or

any of the other functions that are required by many organizations seeking a middleware product.

- Object Monitors - Object monitors are an extension of object request brokers, and they combine the benefits of an ORB with the transaction processing capabilities of distributed transaction processing middleware (Boucher, 1999). The transaction services are handled through a well-defined interface that can be called through the ORB. Because object monitors are a hybrid between ORBs and DTPMs, they combine benefits of both categories. Object monitors allow for encapsulated, reusable objects to communicate with one another through a standard protocol. In addition, they offer extensive transaction control capabilities. Some object monitors also assist in load balancing, failover, thread pooling, and many of the other benefits sought by middleware customers (International Systems Group, 1997).
- Application Servers - Application Servers are perhaps the newest classification of middleware. It is difficult to define what encompasses an application server, because application servers are a fairly new middleware category, and the types of products that are touted to be application servers have a wide range of functionality (Boucher, 1999). Most agree, however, that application servers give users the ability to develop business logic services in a multi-tier environment much more quickly than conventional middleware and with much less programming. Application servers act as a framework for developing these services (Boucher, 1999). Almost all application server packages offer rapid development of middleware services and transaction control. Also, load balancing, failover, security services, thread pooling, transaction queuing, and other functionalities can be found in many of the available products. Some offer CORBA-compliant object support, and some even provide application development tools for creating user interfaces. Some application servers are specifically geared toward internet development, and these are commonly referred to as web application servers (Boucher, 1999). Disadvantages of this type of middleware are that it is still a very confusing segment, with many offerings to choose from. Larger companies are buying out many small companies offering application servers. Because this segment is not well defined and is still in a state of chaos, generalizations cannot be made regarding disadvantages—these must be determined on an individual product basis.
- Enterprise Application Integration Middleware - Enterprise application integration packages are a specialized type of middleware used to integrate legacy applications. This type of software utilizes adapters or connectors, each of which has been specialized to communicate with a specific type of legacy software (Garber, 1999). One clear advantage of using this type of middleware is that it allows legacy integration. Many of these products support the integration of these legacy applications into web applications (Garber, 1999), and businesses can realize huge savings by leveraging existing technology, if they can find an EAI application suitable for their needs. There are disadvantages of using this type of middleware. First, because it is messaging-based, it is difficult to implement transaction control; and second, many of these packages fail to provide the performance features of other middleware types, such as load-balancing, failover, and thread pooling.

EVALUATING MIDDLEWARE ALTERNATIVES FOR INTERNET SYSTEMS

In this section, the information presented above will be used as the basis to propose steps and guidelines that can assist in the selection of appropriate middleware for n-tier internet systems. These steps would be added to the traditional systems development lifecycle phase of systems design. Within the overall development lifecycle, the additional proposed steps are:

1. Gather requirements for the new system (done during systems investigation and analysis)
2. Determine the middleware functionality needed to meet system requirements.
3. Identify middleware categories that offer the necessary functionality.
4. Evaluate and choose middleware products for each of the selected categories.

These steps can be integrated into existing system development processes.

Gather Requirements for the New System

The crucial first step in selecting the appropriate type of middleware is to gather information about the requirements of the new system. If the requirements are not understood and a system is implemented based on incorrect or incomplete requirements, it will not meet the business needs for which it was intended.

The key to performing a thorough analysis is to ask questions and take thorough notes. An exhaustive series of interviews must be performed with all the key personnel involved with the system. There is no magic list of questions or rules of thumb, nor is there a template that can be used to ensure that a requirements analysis is performed thoroughly and to an acceptable level of completeness. However, a set of questions is presented below that will assist in determining the type of middleware that should be used to implement a system:

- Will this system operate on the public Internet or a private Intranet?
- What types of operations will be performed using the system?
- Will e-commerce be performed on the system? Is so, what type of e-commerce will it be—retail sale of tangible goods, digital delivery of goods and services, or electronic commerce among businesses?
- Who will be using the system—customers, employees, or business partners?
- Can it be estimated how many users will concurrently be using the system during peak demand?
- Is the new system replacing an existing system? Why is the existing system being replaced?
- Is there a high degree of growth potential for the new system? Can growth be reasonably predicted and estimated?
- How important is performance?
- How will users interact with data stores on the server?
- Will data stores be read-only or will users perform updates?
- Will transaction control between client and server be necessary?

The above list represents only a small fraction of the questions that would be asked over

the course of a complete and thorough requirements analysis. However, the answers to this set of questions will provide a good insight into the type of system being developed and the requirements of the middleware that should be implemented with the new system. The next step in the process of middleware selection is to determine the types of functionality that will be necessary to meet the requirements of the new system. This step would also be performed as part of the requirements analysis.

System Functionality Determination

This step links each of the questions listed above and will discuss the implications that each answer will have in determining required functionality.

Whether the system will be implemented on the public Internet or a private Intranet was the first question. There are key differences between the two that can help the analyst decide on middleware requirements: security, what types of operations will be performed, whether or not e-commerce will be conducted and what type of e-commerce it will be, who will use the system, the number of users who will be using the system concurrently, what type of legacy system exists, and the degree of performance expected are some of the requirements that must be taken into account. It could be argued that Intranet systems will have a more finite, predictable number of users on the system, and a decreased need for functionalities such as scalability, load-balancing, and failover.

It should also be determined what kinds of operations will be performed on the system. Will this be an interactive e-commerce site that allows users to make secure purchases online? Will this be a business-to-business application that pipelines data from one company to another? Will this be a customer service portal, with username/password security? The answer to these questions can give the analyst a broad understanding of what the system will be expected to do and can lead to more information about security requirements, performance, updateability, and transaction control, to name a few possible functions.

If e-commerce is to be performed, it should be determined what type of e-commerce is taking place: retail sale of tangible goods, digital delivery of goods and services, or electronic commerce among business. For the first category, retail sale of tangible goods, updateability is an important feature because a business would likely want to add and remove products and update prices quickly, as well as to change the graphics and other content to keep the site fresh. Security, transaction control, performance, load-balancing, thread pooling, and failover are important for this type of web site. For digital delivery of goods and services, messaging would likely be added to the previous list of desirable functionality. Business-to-business commerce may need fewer of these functions because they have a more predictable, finite number of users, and they will have a limited, static product line. It is also possible they will us the Internet to transfer data or simply to provide read-only product documentation for customers.

Determining who will use the system can give the analyst insight into which middleware functionality is appropriate. Will retail customers be using the system? Will confidential informa-

tion be transferred via the public Internet? Will only employees be using the system via a private Intranet? Will business partners be using the system? These questions can guide the analyst towards answers regarding security and performance.

The number of concurrent users during peak demand is important. A business-to-consumer e-commerce site, in which case issues such as scalability, load-balancing, failover, and thread pooling will become desirable middleware functions. On the other hand, business-to-business and intranet systems may be able to predict the number of users on the system at one time, in which case these features are less critical.

One important consideration is that the new system may not be replacing an information system, but it may be replacing or enhancing a system that is currently done in part by manual processes. If, for example, the company currently takes phone orders that are entered into a mainframe order-entry application, then perhaps the mainframe order-entry application can be leveraged and used to create a new internet e-commerce system. This type of information is important in helping the analyst to determine whether legacy systems can be leveraged to create the new internet system.

It should also be determined if a high degree of growth is expected and whether or not growth can be reasonably forecast. If the system is not expected to grow rapidly and growth can be accurately forecast, then scalability will be less of an issue. If it is not known how rapidly the system will grow, then scalability will likely be a necessary feature of the new middleware. In the case of a start-up e-commerce site, it may be very difficult to predict usage and growth.

The need for performance should also be investigated. It is hard to imagine that a business would not consider performance, but there could be business-to-business applications that automate overnight data transfers, and this may mean that performance is less of an issue. Failover will be important to ensuring that data transfers have been completed by the next business day. If it is determined that performance is important, then issues such as concurrency control, load balancing, transaction queuing, and message queuing will likely be on the list of desired functionalities.

Another issue that must be addressed is transaction control. What type of transactions will be performed against data servers? Will the users be performing read-only transactions, or will updates to the data servers be necessary? For most e-commerce sites, transaction control will be very important, because customers will be entering order information that must be safely committed to the database. If this is the case, then transaction control will most certainly be added to the list of necessary functions that the new system must perform.

Answers to these questions will help identify necessary middleware functionalities. They will provide a good insight into the system's middleware requirements. Table 2 gives a summary of these findings. It lists the questions that were proposed earlier in this chapter and indicates which common middleware functions will likely be needed based on the answer to each question. The required functionalities will then determine the correct type of middleware to acquire.

Middleware Category Selection

The next step in evaluating middleware alternatives is to review the middleware categories and the common functionality found in each category. This will assist in narrowing down the middleware alternatives that should be investigated further. Table 3 lists the middleware categories discussed in this paper, and indicates which functionalities are commonly found in each category. A "Y" indicates that a middleware category generally possesses the corresponding functionality, while an "N" indicates that it generally does not. Linking functionalities and categories assists developers in identifying what middleware to acquire.

Table 2. Generic Middleware Requirements Analysis Summary

	Advanced Messaging	Application Framework	Dynamic Load Balancing	Encapsulation/Modularization	Legacy Integration	Multi-database Support	Naming Services	Security Services	Thread pooling	Transaction Control	Transaction Queuing	Failover
System Type: Internet		x	x	x	x		x	x	x			x
Intranet												
Type of Operations: E-commerce		x	x	x			x	x	x	x		x
Customer Service				x			x	x				x
Data Exchange								x				x
Type of e-commerce Retail/tangible		x	x	x			x	x	x	x	x	x
Retail/digital		x	x	x			x	x	x	x	x	x
Business-to-business		x		x		x	x	x		x		
User types Customers		x	x	x			x	x	x			x
Employees				x			x	x				
Business partners		x	x	x			x	x	x			x
Concurrency Predictable												
Unpredictable			x						x		x	x
Replacement System Yes					x	x						
No					x							
Growth potential High, unpredictable		x	x	x					x			x
Moderate or low, predictable												
Performance High importance			x						x		x	x
Moderate or low importance												
Transaction Control Multi-phase commit across several heterogeneous databases						x				x	x	
ACID transactions against one or more homogeneous										x	x	
Read-only												

Table 3. Summary of Middleware Functionality

	Middleware Functionality											
Middleware Types	Advanced Messaging	Application Framework	Dynamic Load Balancing	Encapsulation/Modulariza	Legacy Integration	Multi-database Support	Naming Services	Security Services	Thread Pooling	Transaction Control	Transaction Queuing	Failover
Database Access	N	N	N	N	N	Y	N	N	N	N	N	N
Remote Procedure Call	N	N	N	N	N	N	Y	Y	N	N	N	N
Distributed Transaction	N	N	Y	Y	N	Y	N	N	Y	Y	Y	Y
Message-Oriented	Y	N	N	N	N	N	N	N	N	N	N	N
Object Request Broker	Y	N	N	Y	N	N	Y	N	N	N	N	N
Object Monitor	N	N	Y	Y	N	Y	Y	N	Y	Y	Y	Y
Application Server	N	Y	Y	Y	N	N	N	Y	Y	Y	Y	Y
Enterprise Application	Y	N	N	N	Y	N	N	N	N	N	N	N

The next step in this process is to narrow the list of middleware categories to determine those categories of middleware that will best fit the needs of the new system. The analyst may discover that several categories of middleware are suitable. However, it is likely that no category will meet all of the system requirements, and this would necessitate the selection of multiple middleware categories. For many robust e-commerce systems, nearly every type of middleware functionality will be on the final list of requirements, and this will necessitate the use of multiple middleware packages.

It should be reiterated that the application server segment is very promising and should be investigated closely. This segment is very new and still rapidly evolving. However, middleware that falls into this category should be evaluated on a product-by-product basis, because a viable alternative may be found that meets the requirements of the new system.

Middleware Product Selection

The final step in the process of middleware selection is to evaluate individual products and choose the product combination that best meets the needs of the new system. There is not much to say on this topic that is specific and unique to middleware selection or n-tier development. The following list offers suggestions of things to consider when evaluating middleware products:

- Ask vendors for references of organizations currently using the software.

- Ask vendors to identify organizations with systems that have similar functionality to the one being developed by your organization.
- Ask vendors to arrange demonstrations of production systems that are using their product.
- Obtain trial versions of the software and develop a prototype to ensure that the middleware fits the needs of the application.
- Ask the vendors to provide as much assistance as possible in prototype development.
- Discuss training resources so employees can be brought up-to-speed on the use of the new middleware tool.
- Try to select products that are established in the market so that obtaining developers experienced with the product will be possible.
- Ask the vendor what type of technical support package comes with the purchase.
- Consider the financial condition of the middleware source, that is, will the company be around next year to support the product?

This list of considerations can be applied to the purchase of any type of development software and is not exclusive to middleware. Also, it should be stated that there are many other questions that could be asked when making a software purchase. However, the above questions can help a business select the right vendor and product to provide the middleware to use in building an n-tier system.

SUMMARY

Two-tier technologies are inadequate for large-scale e-commerce and business systems. Businesses must use three- or n-tier architectures to address the challenges presented when implementing these kinds of systems. The enabling technology that makes this possible is middleware.

To ensure the success of a three- or n-tier system, businesses must understand the system being built and the technology being used to build it. Key steps to help ensure success are:

1. Understand the requirements of the new system.
2. Understand the functionality needed to fulfill the requirements.
3. Understand the various categories of middleware and the functionality that is typical of each category.
4. Evaluate products from those categories identified as meeting the needs of the new system.

Following these steps will help to ease the pain of middleware selection and will facilitate the successful implementation of new e-commerce systems for businesses that wisely choose to use an n-tier architecture.

REFERENCES

Bernstein, Philip A. (1996). Middleware: A Model for Distributed System Services. *Communication of the ACM, 39(2),* 86-98.

Bielski, Lauren (January, 1999). Ready for Multi-Tier, Distributed Computing? *American Bankers Association Journal, 91(1),* 53-55.

Boucher, Karen (December 1999). Application Servers on the Front Lines. *Software Magazine, 19(3),* 46-51.

Dogac, Dengi, and Oszu (1998). Distributed Object Computing Platforms. *Communications of the ACM, 1998, 41(9),* 95-103.

Eckerson, Wayne W (1995). Three-Tier Client/Server Architecture. *Open Information Systems, 10(1),* 1-20.

Edwards, Harkey, and Orfali (1999). *Client/Server Survival Guide, Third Edition.* New York: John Wiley and Sons, Inc.

Edwards, Jeri (1999). *Three-Tier Client/Server at Work.* New York: John Wiley and Sons, Inc.

Garber, Lee (1999). Middleware Moves to the Forefront. *Computer, 32(5),* 17-19.

Gaudin, Sharon (1997). IBM Middleware Befriends CORBA. *Computerworld, 31(20),* 41-42.

Hoffman, Richard (1999). No More Middle Ground for Middleware. *Network Computing, 10(10),* 76-77.

International Systems Group (1997). *Middleware – The Essential Component for Enterprise Client/Server Applications.* International Systems Group Incorporated.

Lewis, Ted (1998). The Legacy Maturity Model. *Computer, 31(11),* 125-127.

McFall, Cynthia (1998). An Object Infrastructure for Internet Middleware. *IEEE Internet Computing, 2(2),* 46-51.

Sharon, Dave (1997). Avoiding a Middleware Muddle. *IEEE Software, 14(6),* 92-98.