The Middleware Muddle[1]

Application servers and TP monitors are finding new life on the Net.

David Ritter
dhr@webmine.com

A new menagerie of middleware is emerging. These products promise great flexibility in partitioning enterprise applications across the diverse corporate computing landscape. What factors should you consider when choosing a solution, and how do current products stack up? More important to the focus of this article, what role should Web servers play?

What's What

These days, "middleware" describes any software component that sits between application users at their PCs and the RDBMS or legacy system that directly manages underlying data. This term, like many others in the field, is applied so broadly that it's lost its meaning. To help sort things out, I'll propose some more specific categories. Still, these things are rarely clear cut — the functionality of a particular product may cut across several layers.

I've ordered the categories from simple to complex. Some middleware just provides a mechanism for data to get from place to place. More sophisticated offerings help manage application logic and resources. The most robust tools directly support significant application functionality, such as credit card transactions for e-commerce. This article focuses on the more complete offerings, but I'll also offer some background on the basic plumbing.

Database Transparency

If you're dealing with multiple database systems, a common access API is almost essential. This allows the use of standard tools and greatly simplifies the application development process. The most visible examples of APIs for database transparency are Microsoft's ODBC, OLE DB, and ActiveX Data Object (ADO) interfaces. For Java developers, JDBC is gaining acceptance as a common database access interface. In the C++ world, RogueWave Software's DBTools++ has a wide following.

IPC and Objects

These protocols and products facilitate interprocess communication (IPC) and object distribution. They form the basic glue that holds multitier applications together. Most of the higher-level products I'll discuss use one or more of these underlying protocols. The key players are:

- **Remote Procedure Call (RPC) and its Java equivalent, Remote Method Invocation (RMI)**. Essentially, these protocols allow your application to call functions and pass parameters across process and machine boundaries. They're usually synchronous, meaning that each operation is completed before the next operation begins. These services are provided by the operating system or language development environment. RPC is usually based on the Distributed Computing Environment (DCE) infrastructure.
- **Messaging systems**. In contrast, messaging systems are typically asynchronous. Requests for services are queued and processed according to priority and the resource availability, and responses are returned to the requester to indicate the success or failure of the operation. They're often used for workflow and process-control applications, as well as for WAN applications with slower, less reliable connections.
- **Distributed object systems**. Object systems provide facilities for locating and interacting with objects in a distributed environment. Objects are identified by name or by the services and interfaces they support. The implementation of the object and the platform on which it runs are transparent to the client.

---

[1] This article originally appeared in the May 1998 issue of *DBMS*. It is reprinted with the permission of the publisher, Miller Freeman, Inc. Recently, *DBMS* and *Database Programming and Design* magazines merged into a new publication, entitled *Intelligent Enterprise* (http://www.intelligententerprise.com).

War rages in this area. On the surface, it's Microsoft's Distributed COM and ActiveX technologies vs. the Object Management Group's combination of CORBA, the Internet Inter-ORB Protocol (IIOP), and JavaBeans. More deeply, this battle reflects the struggle between the openness, maturity, and scalability of Unix and the growing Microsoft NT juggernaut. Interoperability between these standards is emerging slowly, but it has yet to gain the confidence of either camp. What's your religion?

Objects are the real currency of modern multitiered applications. All higher-level products discussed here focus on the management of objects in ever more complete ways. Planning your enterprise strategy in terms of objects and components will allow you to best leverage this rapidly emerging technology. Judith Hurwitz provided a round-up of products at this layer in her DBMS article, "Sorting Out Middleware" (January 1998, page 10).

Transaction Processing Monitors

Simply put, heavy access to shared resources leads to bottlenecks that prevent work from getting done. Many early forays into client/server computing at the enterprise level fell flat as the result of inadequate database resource management. Early attempts to use RDBMSs to drive dynamic content on the Web met a similar fate for the same reasons. In many cases, it wasn't the actual processing of the SQL statements that caused the problem. The slowdowns resulted from inadequate database connection management and ineffective approaches to caching.

Beginning with IBM's Customer Information Control System (CICS, pronounced "kicks" by war-torn veterans) in the early 1970s, systems have been developed to provide database resource and transaction management for applications. The success of these products is clearly demonstrated by the fact that of the top 20 TPC-C benchmark results (ranked by transactions per minute as of February 2, 1998), every single test environment included a database middleware technology. If the same results are ranked by price/performance, 18 of the top 20 used a TP monitor (sources: www.tpc.org and tuxedo.novell.com/action/tpc.htm).

TP monitors evolved out of these needs and others:

- Many organizations use more than one database system, and business needs call for transactions to be performed that span across them.
- Many database systems require an entire operating system process per connected user. For applications with hundreds of users, even the largest hosts are overwhelmed.
- Establishing a connection to the database is frequently slow. With many users connecting and disconnecting frequently, system performance degrades severely.
- TP monitors attack these problems by:
- Offering connectivity to a variety of different database systems simultaneously.
- Providing a two-phase commit protocol, which guarantees the completeness of database transactions across multiple databases.
- Processing user requests using lightweight operating system threads, rather than full processes. This allows TP monitors to exploit today's SMP systems, such as the Sun Enterprise, Digital Alpha, and Compaq Proliant.
- Maintaining a persistent pool of database connections and sharing them across users. In most applications, each user is actually accessing the database for only a fraction of their total time online. Often, hundreds of "simultaneous" users can be efficiently served by one-third or even one-tenth the number of database connections required for direct access.
- Keeping shared database connections open over long periods, dramatically reducing the amount of connection traffic.
- Load balancing, which involves measuring the usage of shared resources and directing requests to the least-used servers. Monitors can also detect and act on situations where a server or other resource has failed and needs to be restarted.
- Managing requests asynchronously, distributing multiple requests over separate database connections to the same server (known as pipeline parallelism).

- Distributing requests over multiple database servers. This technique is known as fan-out parallelism.
- Communicating peer-to-peer with other transaction processing monitors to coordinate the operation of partitioned and distributed applications.

The advent of TP monitors has led to some interesting questions about how database products are licensed. Most vendors, including Oracle, Sybase, and Informix, license their products on the basis of the number of connections. The more simultaneous connections to the database, the more you pay. But with a TP monitor, many users may share a small number of connections. Does this mean you can get a bargain on your RDBMS license? Probably not — the major vendors now take this into account and require pricing to be based on the number of actual users connecting, regardless of whether they use middleware.

This discrepancy is especially painful for Web-based applications, where there are often millions of users. In many cases, the interaction of these users with the database is very limited. For example, a Web site that uses a database only for user registration might need only 10 pooled database connections to serve the entire population. Still, the RDBMS vendor will typically require a license to support an unlimited number of users. On Unix systems, these licenses can run hundreds of thousands of dollars. This pricing consideration is helping drive Web applications to Windows NT, and it will dramatically affect the margins and platform considerations for RDBMS vendors in the coming months and years.

In the open systems arena, BEA Systems' Tuxedo leads the TP monitor category. Spun out from Novell in February 1996, this robust and mature TP monitor product was used in approximately 80 percent of the TPC-C results mentioned earlier. It was the 1997 DBMS Readers' Choice winner for best TP monitor, and it's also incorporated into major turnkey application offerings such as PeopleSoft. Other significant TP monitor products include:

- Transarc Encina (1996 DBMS Readers' Choice winner)
- IBM Transaction Server products (includes CICS and Encina)
- Digital ACMSxp
- NCR (distributed by Entersoft)
- Microsoft Transaction Server (MTS)

Object Integration

Some TP monitors deal effectively with objects. Microsoft Transaction Server is notable in this respect. It has strong integration with DCOM; essentially, any ActiveX object can be managed and cached by MTS. This makes application partitioning much more straightforward because existing components built with Visual Basic, C++, or J++ can be easily relocated to the server. MTS keeps the object "alive" for reuse, eliminating the need to re-create instances continually to service new requests. MTS is (for obvious reasons) bound tightly to Windows NT, but its robustness and low cost make it very attractive. Microsoft has done an excellent job of integrating Java and ActiveX — all Java objects are automatically exposed. This eases the integration of MTS into a multiplatform environment.

On the Unix side, the Object Transaction Server (OTS) specification from the OMG is intended to unify TP monitor functionality with object request brokers. This extension of the CORBA protocol is reflected in JavaSoft's Java Transaction Service specification, which made its commercial debut in February with the Sybase Jaguar Component Transaction Server.

Another emerging standard to be aware of when evaluating TP monitors and application servers is the Transaction Architecture (XA), a specification developed by The Open Group (www.opengroup.org). XA defines the interface between a transaction manager and the resources it uses, such as database systems and communication channels. Most major Unix TP monitors and database systems support this standard. Windows NT support lags but is reported to be on the way for MTS.

You also should consider the underlying communication protocol used by the application server. Servers that are strictly tied to RPC (or DCOM in the case of MTS) may be limited by the synchronous nature of the plumbing. Products that allow the use of queued messages may offer more flexibility, especially over a WAN. For example, Encina can use IBM's MQSeries middleware. Microsoft has indicated that integration of MTS and its Message Queuing Server will happen in later versions.

Application Servers

If TP monitors do all that, what's left?

While TP monitors are effective tools for large-scale OLTP, they generally don't have facilities for:

- Hosting and managing application logic
- Locating and instantiating objects (name services)
- Caching objects and controlling object lifetimes.

Developing multitiered applications is all about partitioning, which is the division of labor between the client and one or more specialized servers. Application servers facilitate partitioning by allowing the components of the application to be moved to the architectural level and physical platform where they can work most efficiently. Need to run a forecast against a 10GB database without having sales operations grind to a halt? Implement a component that runs in the middle tier on a system with lots of CPU and memory but not much disk. Need to turn the results into nice graphs? Let your hundreds of PCs do that work in a really distributed fashion, on each user's desktop.

An application server supports the definition of application logic. Some servers provide their own development tools and languages for this purpose. Others rely on object standards such as CORBA, COM, or JavaBeans. In almost all cases, the application logic needs to be encapsulated as one or more objects. These objects expose their capabilities through methods that can be invoked by the application code directly or by other objects.

Given the flexibility of modern RDBMSs, it's tempting to put significant amounts of application logic directly into the database in the form of stored procedures. Except for basic referential and data integrity constraints, this is generally a bad idea. For most applications, the database server throughput is the most precious resource. If the server is performing an application-specific task for one resource-hungry user, it's unavailable to serve the needs of all other users. It's better to partition this work so hardware resources can focus more accurately on the neediest activities.

Until recently, most application servers were tightly bundled into enterprise application development platforms. Forté Software provides a leading example. The Forté product line includes complete GUI development tools, database connectivity components, and deployment facilities. All of these are built around a robust application server. (See Figure 1, page 48.) The designer creates the application as though it were to run all on one machine, and then it uses Forté's partitioning tools to divide the work between the client and application servers. Unify Corp.'s Vision also follows this model.

Until the end of 1995, this class of high-end enterprise tools was a fairly well-understood and mature category. Then the Web happened.

Plain Old Web Servers Aren't So Plain Anymore

Web servers are for Web sites, right? Yes, but increasingly, they also serve a more general role in the enterprise. The new generation of application server products is heavily biased toward the Web. They leverage Web protocols such as HTML and HTTP, and they use the Web browser as the primary client environment.

The definition of the Web server platform has changed dramatically in the last 18 months. This is being driven by the fierce growth of the Web as an information and commerce platform. While good numbers are still hard to come by, anecdotal return on investment analysis of intranets shows costs savings of many millions of dollars, especially in areas such as sales and customer support. Many businesses expect Web commerce to make up an increasing percentage of their overall sales. For example, John Chambers, CEO of Cisco Systems, said that his company's Internet commerce and intranet systems are saving the company more than $250 million each year. Their Web site handles approximately 40 percent of their sales, for a projected total of $3 billion in Web sales in 1998. (Source: John Chambers at Comdex keynote speech in November 1997.) SAP responded to this trend by joining Intel in a venture (named Pandesic) to bring their enterprise applications to the Web.

On the technical side, this growth is facilitated by a critical mass of standards, such as HTTP, SSL, and Java. These standards are embodied in the Web server, which acts as a focal point for the integration of new tools and technologies. Table 1, shows this remarkable progression of functionality.

These products are only examples. The range of vendors and tools that enable the Web as an application platform increases daily, and the existing products continue to improve. Along with this explosion on the server, the capabilities of the Web client have improved in parallel. With the growing maturity of client-side JavaScript, Java run times, Dynamic HTML, and ActiveX, Web applications need no longer suffer from the primitive, limited user interface presented by older HTML offerings.

The key players in this new space come at the market from different angles. From the RDBMS vendors, we have database-centric application servers such as Oracle Web Application Server and Sybase Jaguar CTS. From the Internet and e-commerce fields come servers focused on integration with Web servers, EDI, and electronic payment systems. Leading products in this field include Kiva (recently acquired by Netscape), InterWorld Commerce Exchange, Dynamo from Art Technology Group, and Novera EPIC.

Some of the new offerings draw heavily on the Forté example, although none offers the same level of seamless partitioning support. They do incorporate their own development tools and proprietary technologies in an effort to present a complete, fully integrated solution. Progress Software's WebSpeed (reviewed in the November 1997 issue of DBMS, page 33) offers transaction services, tight integration with the Progress database, wizards and templates, and a proprietary scripting language. It opens the door for the use of Java for application components and access to any ODBC-compliant database.

SilverStream (reviewed in the March 1998 issue of DBMS, page 29) was developed by many of the same people who brought us PowerBuilder, but it is targeted squarely at Web applications. It offers a slick set of user interface, object, and database design tools. The heart and soul of PowerBuilder, the DataWindow, has been translated very well into the new environment. The scripting language is Java, so the learning curve should be short for most developers. The application server manages the execution of agents, which can run in response to database updates, at scheduled times, or at the request of application code. Agents can process data sets retrieved from the database on a row-by-row basis to provide procedural filtering.

Because it tries to do so much, the initial release of SilverStream comes up short in some areas. All its application components are stored in the database in an opaque format, reducing the flexibility and openness of the environment. External Java class libraries aren't easily incorporated. The user interface class libraries don't conform to any of the AWT, AFC, or JFC standards. The application server lacks fault-tolerance features, and the absence of a script debugger is very problematic. Given SilverStream's depth of experience and momentum, it's reasonable to expect that the product will improve rapidly.

Upsides and Downsides

With all of these products, Java emerges as a clear winner. All products offer Java compatibility at some level. Many, such as Dynamo and Novera EPIC, are based on Java from the ground up. As noted earlier, even Microsoft has good Java support. If you're searching for the development language that will go the farthest to span multiple platforms, promote reuse, and operate effectively on both the client and server tiers of your architecture, Java is by far the best bet. Implementations are maturing rapidly, and early performance problems are quickly yielding to better run times and faster CPUs.

Beans are to Java what ActiveX is to COM. They provide a model for components, including event handling. Until recently, the JavaBeans model has been most commonly used to package visual controls for client applications. But the next major step in Java's drive to become an enterprise standard comes with JavaSoft's introduction of Enterprise JavaBeans (EJB). This extension focuses on specialized, nonvisual Beans for the server. The Enterprise JavaBean execution environment provides thread pooling and implicit transaction management. (See Figure 2.) An impressive list of TP monitor and application server vendors have indicated support for this component model. Some will have products ready to ship when the EJB specification is finalized. Unfortunately, Microsoft is currently absent from the list. Still, the wide adoption of EJBs promises to enable the broad reuse of application components.

What's the downside to using a Web server as a tier in an enterprise application architecture? There are several issues:

- HTTP isn't ideally suited for client/server applications. It's stateless, meaning that there's no persistent connection to the server. This means that state information has to be included in every request, generally at a significant performance penalty.
- The frantic pace of development has led to some shaky product releases with questionable quality.
- The reliability and performance of JavaScript and Java executing within the Web browser are still poor. This has led Web application server vendors such as SilverStream to offer dedicated client run times for executing applications. This solution is reasonable, but it defeats one major promise of the Web platform, namely, not needing to install custom software on the client.
- Dynamic HTML (DHTML) and the user-interface class libraries for Java development aren't standard yet. If your application requires any advanced user interface features, you'll probably need to restrict your users to a specific browser vendor and version.
- Within the Web browser, Java is "sandboxed." Without permission from the user, downloaded Java can't write to the PC's disk or perform other potentially destructive actions.

Choose Your Weapons

For today's enterprise applications, flexible resource use is a key to success. Building component-based applications will let you use the latest application server technology to meet these challenges. These products have finally matured to the point that it's no longer necessary to build your own glue, transport mechanisms, communication protocols, or database connectivity, even for very demanding applications.

For the most resource-intensive applications, dedicated TP monitors offer the highest performance. For applications involving more complex rules and logic, more general application servers provide an excellent framework. In any case, the bulk of the new technology is converging around Web standards. Developing in Java will open a wide range of choices for partitioning and deploying your applications. In the Unix world, there is a rich set of products based around CORBA and Java. For Windows NT, expect Microsoft's MTS and its supporting cast to dominate. Selecting a vendor with a strong commitment to standards and open architecture is the best insurance in this time of rapid change.

David Ritter is the principal of WebMine, a Boston-based consulting practice focused on information and technology solutions for Internet business. He was previously the vice president of engineering at Firefly Network and managed the development of OLAP client products for Oracle Corp. He has spoken widely on topics ranging from data warehousing to Internet privacy. You can email David at dhr@webmine.com.

Web Platform Features	Capabilities
Common Gateway Interface (CGI)	Handles full requests with separate server processes; slow and difficult to program; no built-in database support.
Server APIs: Netscape NSAPI; Microsoft ISAPI	Handles full requests with in-process libraries; much faster, but even more difficult to program; no built-in database support.
First-generation scripting environments: Netscape LiveWire; Microsoft Active Server Pages (ASP); Apache with Java Servlets; ColdFusion; WebObjects	Integrated with HTML, allowing incremental extensions to pages; high-level scripting in JavaScript, VB Script, or Java speeds development; some performance problems due to script execution; some object and database support; very limited security.
Second-generation application environments: LiveWire plus LiveConnect and Borland/Visigenic ORB; ASP plus ADO and MTS; Apache with CORBA and JavaBeans; SilverStream 1.0; Progress Software WebSpeed	More generalized object support; Java and CORBA integration; database connection pooling; better caching and precompilation improves performance; better security with 40-bit SSL.
Third-generation enterprise and commerce environments: Netscape with Kiva Application Server and Actra commerce servers; Microsoft Site Server 3.0 with Active User Object, MTS 2.0, and Merchant Server; Open Market LiveCommerce and Transact	Sophisticated database resource pooling and object caching; integration with knowledge and content servers; extends into more vertical application areas, such as user registration and profiling; high-end user-to-business and business-to-business commerce functionality; integration with back-end payment systems; more extensive security with 128-bit SSL and digital certificates.

Table 1. The progression of Web platform functionality.

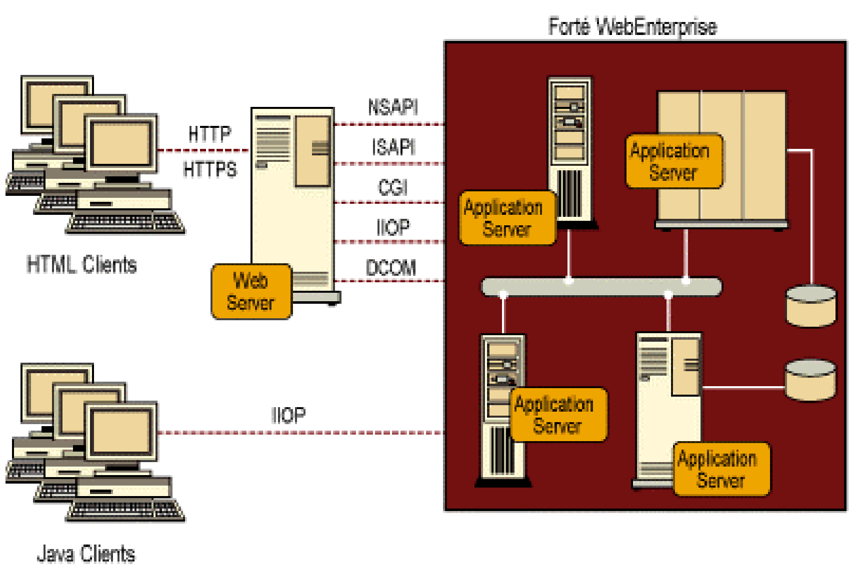

Figure 1. The Forté WebEnterprise architecture. Courtesy of Forté Software Inc.

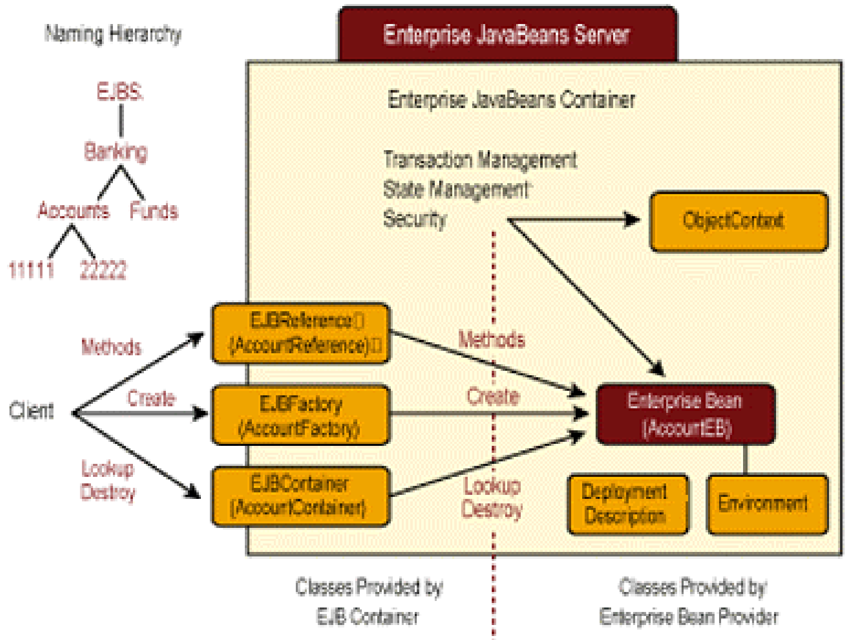

Figure 2. The Enterprise JavaBean execution environment. Courtesy of JavaSoft division of Sun Microsystems Inc.