4 Evolution of Application Servers

As distributed object applications become mainstream, and Web-based business takes hold, enterprises are shifting to the middle tier. By placing business logic on middle-tier servers, enterprises can easily update applications for thousands of clients—without requiring clients to perform installations.

However, middle-tier architectures are much more difficult to develop than traditional client/server applications, and pose a set of unique challenges for deployment and management.

For enterprises to make the leap to multi-tier, distributed architectures, mainstream IT developers must be able to develop sophisticated distributed applications using a visual environment. IT must be able to easily deploy these applications in a test and production environment that is shared with administrators. And administrators need to manage these de-centralized applications from a central location.

What enterprises need is a solution that integrates the key technologies necessary to simplify the development, deployment, and management of distributed, multi-tier applications. The solution is an application server. There are currently two types of application servers on the market: Web application servers and legacy application servers.

Web application servers enable easy development of Internet applications, and ***Legacy application servers*** rely on existing systems such as TP Monitors. These types of application servers are fully described later in the paper. However, neither of these application servers fully meet enterprise requirements for the Web era.

Borland delivers the next generation ***Enterprise application server***—an integrated solution that combines the best of application server technology to meet the needs of enterprises as they shift to multi-tier, distributed, Web-based solutions.

4.1 Enterprises Shift to the Middle-tier

Enterprises are shifting to middle-tier servers in droves as distributed applications become mainstream. Distributed object applications offer enterprises improved fault

tolerance and scalability, faster time-to-market by reusing application components, and a way to tie disparate heterogeneous environments together using the power of the Web.
By partitioning applications into components, and shifting business logic onto middle-tier servers, IT can vastly reduce turnaround time for development of applications by reusing the logic in these middle-tier server objects. Another key benefit of middle-tier architectures is the ability to locate business logic on the server rather than the client. sssThis means that if code needs to change, it can be changed in one place rather than in each of 10,000 client applications.
Multi-tier distributed object architectures also offer significant advantages for enterprises as they seek to integrate new Web-based business with existing systems. Web-based business applications require a thin-client architecture to support browser-based clients. These clients need to interact with intranet-based resources, but often have limited system resources that make it difficult to download applets.
Using server-side, middle-tier solutions, IT can alleviate the burden from these clients by transferring the bulk of business logic to middle tier servers-leaving clients with a thin interface to the rich functionality users craves.
These server-side solutions also bridge heterogeneous platforms and integrate with legacy systems, enabling Web-based applications to integrate with existing enterprise systems.

4.2 Challenges Posed by Multi-tier, Distributed Applications

Although multi-tier, distributed architectures offer numerous advantages to enterprises, the shift to this new architecture comes with its own unique growing pains:
Developing a multi-tier, distributed application is complex deploying the hundreds of components that make up a distributed application is challenging managing thousands of distributed components is daunting.

4.3 Developing a Multi-tier, Distributed Application Is Complex

Highly technical, advanced programmers have been developing multi-tier, distributed applications for some time. But these developers have deep knowledge of underlying system-level complexities such as concurrency, locking, transaction management,

security, and scalability. They also understand how to manage access to system resources such as threads, memory, database connections, and network connections.

But the complexity of these issues has limited the spread of multi-tier development because mainstream IT developers typically have more experience with business logic development rather than system-level programming.

4.3.1 Deploying the Hundreds of Components that Make Up a Distributed Application Is Challenging

Most distributed applications are comprised of hundreds of components. Each component has properties that must be configured to determine how it starts up, if it logs information, and so on. Many times, the way these properties are set depends on the platform where the component resides.

Furthermore, the test and production environments for these distributed applications are typically disconnected, preventing developers from making intelligent modifications to these applications to improve performance and scalability in real-time deployments. Once these applications are deployed, keeping track of thousands of decentralized objects is challenging.

4.3.2 Managing Thousands of Distributed Components Is Daunting

Once distributed applications are deployed, administrators need to ensure that they keep running. Components of distributed applications can reside anywhere across the enterprise, and experience failures from all manner of causes including system failure, network interruptions, and application errors.

Administrators must quickly discover that a component has failed, and then take immediate steps to restart the component, or to start a replica of the component and redirect client traffic.

However network and system management, and particularly SNMP, all approach management from the point of view of a host. Hosts perform particular functions, have

software running on them, supports SNMP MIBs, and so on. But a distributed application does not run on a host—it runs on an interconnected network of hosts. By looking at a single host, it is impossible to tell whether an application is running.
To administer a distributed application, administrators must administer the entire network— this is a daunting task if you do not have the right tools.

4.3.3 An Enterprise Application Server Is the Answer

Enterprises that are moving to multi-tier, distributed environments need an application server. An application server ties together disparate technologies to make development, deployment, and management of multi-tier, distributed applications easier. Currently, there are a plethora of "application server" technologies on the market. These technologies break down into two basic categories:

Web application server
Legacy application server

4.3.3.1 Web Application Server

The Web application server provides a development environment for Web-based HTML applications. It enables HTML authoring of Web-based client/server applications. In this architecture, a Web application server resides on the Web server, and handles incoming client requests. ODBC and JDBC are used to connect with databases.
This type of application server is typically easy to use and Java-oriented with support for Enterprise JavaBeans for server-side component development.
However, these solutions do not meet the requirements of mission-critical enterprise applications-they do not provide support for transactions, they provide little security, they do not access legacy systems, and they typically have less than optimal performance.
Examples of a Web application server are solutions provided by Netscape, NetDynamics, and WebLogic.

4.3.3.2 Legacy Application Server

The legacy application server provides a business-critical application infrastructure with transactions handling, security, failover, and load balancing to integrate data in existing, legacy systems such as TP Monitors. However, many of these solutions bootstrap distributed objects into client/server architecture, and thereby provide cobbled support for Web-based activities.

4.3.4 Enterprise Application Server

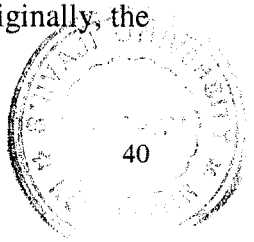
Most existing application servers do not provide a complete solution. What enterprises need is the best of both worlds. They need an application server built on an enterprise-class infrastructure that provides transactions, security, and centralized management for Web-based distributed applications.

One line of development begins with Web site development tools.

The first Web applications were built with handmade tools. They were developed as needed and written in C, C++, Perl, or whatever else happened to be available. These handmade tools eventually gave way to the first integrated Web development tools. Integrated tools sported fancy user interfaces, streamlined the generation of HTML, and added rudimentary back-end integration. The heirs to this line still maintain the family resemblance. They tend to provide a rich development environment and generate high-quality, dynamic user interfaces but provide weaker support for enterprise application integration.

The other line grew from n-tier application architectures.

N-tier architectures are a natural evolution of the client/server paradigm. Three-tier architectures consist of a back-end system (or systems), a user interface, and a middle-tier consisting of a server executing the business logic. The middle tier was responsible for integrating the data provided by the back-end systems into a single whole. Originally, the



middle-tier software was written (and rewritten) by hand. It handled security, interfacing, and a host of other (fairly generic) responsibilities for itself. Successors to these early tools unloaded most of these generic responsibilities onto a standard platform and occasionally provided tools to aid in the generation of the business logic. Once again, the family resemblance is still apparent. The heirs tend to excel at back-end integration while being weaker on presentation.

It is likely that what we now call application servers could have arisen from either of these two separate lines of development, since both strove to solve the same problem -- that of creating effective distributed applications -- for different reasons. Critical mass developed, however, when the two lines met, and the resulting explosion created a market in which 30-plus companies claim to be players.

A third line of application server has appeared recently. Often springing do novo from very large, established companies, this line is the result of the identification of application servers as a strategic trend in the marketplace and the subsequent introduction of products that target that market. IBM's WebSphere is perhaps the best example of this line. Sun could have taken the same route and developed and introduced its own product, but opted instead to acquire NetDynamics. Microsoft is still without a clear product in this market, but may eventually provide a competitor.

An application server, then, is a middle-tier application that combines three components: pieces for communicating with back-end systems; pieces for communicating with front-end clients (often, but not necessarily Web clients); and a framework upon which business logic can be hung. The result is a system that is modular, highly scalable, robust, and dynamic enough to meet the needs of today's businesses.

As it turns out, Java technology plays nicely in this space. An application server based on Sun's enterprise APIs takes these trends to their logical conclusion -- a common, ubiquitous, platform-independent substrate upon which enterprise applications can be built. The open nature of the APIs enables developers to select from best-of-breed

solutions without fear of vendor lock-in. It also opens the door for component-based application development, with emphasis on the server side. This is a trend that has never quite materialized but holds great promise for accelerating the speed at which quality applications can be written.

4.3.4.1 Purpose and use

At this point it's worth taking a look at application servers in terms of both their functional role and their place in an n-tier architecture.

Functionally, application servers host business logic and consolidate the raw information provided by back-end systems and databases into a form that has real business meaning -- hence the term "application" in application server. This used to mean code written to a proprietary API. It now means Java code written to the Enterprise JavaBeans (EJB) specification.

Application servers increasingly are responsible for providing adapters for communicating with external systems, including legacy systems and applications from vendors like PeopleSoft and SAP. They also provide tools for dynamic HTML generation and client communication.

Architecturally, application servers reside on the middle tiers of an n-tier architecture.

4.3.4.2 Trends and expectations

Three major trends currently seem to shape the application server space.

First, all vendors seem to be committed to supporting the Java 2 platform and Sun's enterprise APIs. Many players, however, are taking a conservative approach -- they are committed but intend to tailor implementation to the level of demand they see from their customers. Therefore, expect to see companies adopt the technology in a piecemeal fashion.

Second, the products are converging in terms of the features they support. With Sun's enterprise APIs as a catalyst and customer needs usually spanning the range from back to front, application servers increasingly will provide one-stop shopping. This will force vendors to compete in terms of stability, security, and performance instead of proprietary features.

Third, expect the field to make more sense in a year's time. As I write this, there are 30-plus (perhaps even 40-plus) vendors who claim to provide "application servers" (and many such products showcased at the Java Business Expo conference) -- each dramatically different from the next. The field is full and the consolidation of features mentioned above should reduce the ranks of vendors, not increase it. The suite of features that application servers provide also should become more standard, allowing any application written to Enterprise JavaBeans to run on any number of app servers supporting it -- extending Write Once, Run Anywhere to the enterprise middleware arena.

So much has been written and said about application servers that many knowledgeable people within the computer industry are wondering what they really are.

The seven key questions when you build an application server:

1. Which distributed computing model?
2. Which development language?
3. Which platform?
4. How will we achieve scalability across one or more systems?
5. What services will our application server environment need?
6. How will we achieve mainframe connectivity?
7. How can we guarantee transaction integrity?