Java Based Application Servers for the Financial Sector

by: Anura Gurugé

Once Y2K becomes but history, the most pressing challenge facing MIS professionals in the financial sector will be that of persuasively leveraging the popularity and the reach of the Web to enhance corporate revenues, market share and profitability. Indubitably, the Web will thus have to be gainfully harnessed to dramatically increase corporate earnings via electronic commerce transactions, expedite customer interactions via automated 'Internet Call Centers', and in general enhance overall competitiveness through new, innovative electronic services [e.g. home-banking, 'instantaneous' loan approvals, etc.] Though Web based, on-line investing has only been around since c.1996, and security concerns still abound, a May 1999 report from Dataquest shows that 15 million people in North America alone were using the Web on a regular basis to track their investment portfolios, while 4.5 million were already buying and selling shares and mutual funds on-line. Charles Schwab, one of the pioneers in Web based trading, claims that it has 2.2 million online customers – and that 61% of their trades, totaling nearly $7B a week, are now conducted securely and expeditiously over the Internet without any intervention by customer service representatives or brokers. Web-based trading, Home Banking and Internet Call Centers by 2001 will be indelible, integral and strategic features of the financial sector landscape.

Established, successful financial institutes will invariably discover that they do not have the luxury of time, resources, or latitude to develop brand new systems, from scratch, just to address the burgeoning Web opportunities. Instead, they will have no choice but to extend their existing, highly proven information systems, replete with mission-critical applications and diverse databases, to incorporate Web-based transactions. Java Application Servers are rapidly gaining a stellar reputation as very powerful, flexible, strategic, cost-effective, and platform-independent means of synergistically integrating existing information systems with the Web. Java Application Servers, in essence, enable existing information systems, applications, databases and processes to be reused in the context of the Web.

What is a Java Application Server?

A Java Application Server is a server resident, middleware component. It is written and developed using the highly acclaimed and now widely popular Java programming language that was introduced by Sun Microsystems, in 1995, to explicitly address the needs of Web oriented applications and the creation of truly interactive Web pages. A Java Application Server sits between a standard Web Server and the existing information systems – where the existing information systems are likely to consist of a mix of mainframes, minicomputers, Unix systems and NT servers.

Java Application Servers utilize a classic 3-tier architecture where the existing information systems act as data servers and the clients are PC or workstation users connected to the Web or an intranet. Within this 3-tier configuration the Java Application Server, true to its name, acts as the 'application server component' between the data servers and the clients. **Fig. 1** shows the 3-tier architecture of a Java Application Server.

Java Application Servers will enable financial institutions to readily and cost-effectively implement a new genre of object-oriented applications specifically targeted at Web users. These new, Web-centric applications, however, will be able to adroitly access, manipulate, synthesize and above all reuse any and all forms of existing corporate data and business logic – irrespective of their nature, location, or vintage. This 'reusability' aspect is the primary lure and justification for this type of application server vis-à-vis Web integration. These application servers enable corporations to maximize and extrapolate the significant investments that they already have in terms of computing platforms, mission-critical applications and databases to address their emerging Web requirements – obviating the need to recreate or replicate resources purely for Web use.

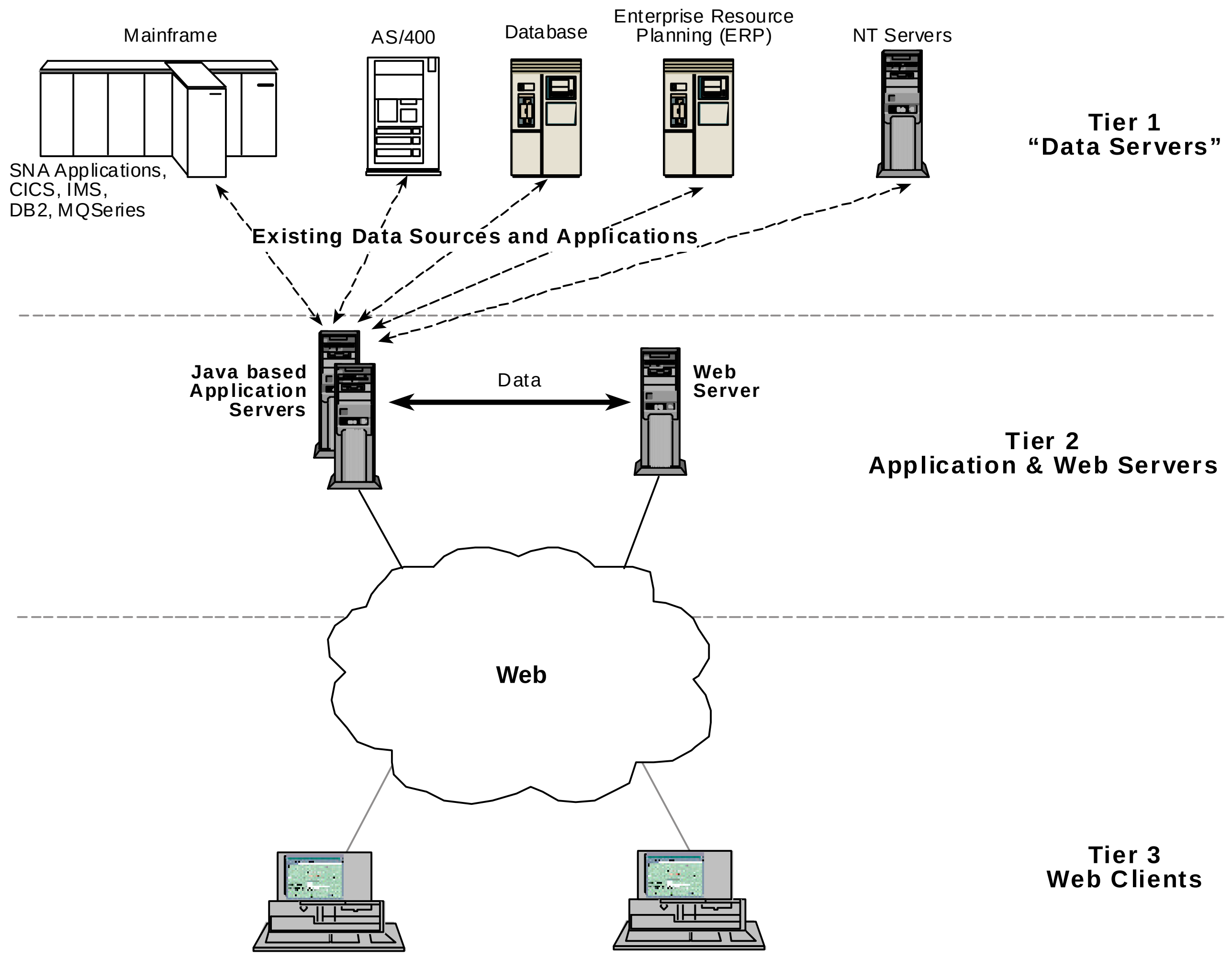
Mainframe
AS/400
Database
Enterprise Resource Planning (ERP)
NT Servers
Tier 1 "Data Servers"
SNA Applications, CICS, IMS, DB2, MQSeries
Existing Data Sources and Applications
Java based Application Servers
Data
Web Server
Tier 2 Application & Web Servers
Web
Tier 3 Web Clients

Fig. 1: The 3-Tier Client-Server Architecture of Java based Application Servers

Feature rich and proven Java Application Servers are now readily available from a variety of different companies – including some big names such as Netscape and Sun. Some of the better known Java Application Servers now available include: BEA's WebLogic, Sun's NetDynamics, Bluestone's Sapphire/Web, Novera's jBuiness, Netscape's Application Server and Inprise's Application Server.

Why Java?

Middleware solutions that enable existing Information Systems to be extended to incorporate Web based applications do not necessarily have to be Java based – and there are indeed many non-Java application server offerings on the market today with IBM's well known MQseries family being but one example. There are, however, definite tangible advantages for opting for a Java Application Server. Key among these advantages being:

1. **Platform Independence**: 'Write once; run anywhere' is Java's overriding and now substantiated value-proposition. Consequently a Java Application Server is not tied to a

particular computing platform such as Unix or NT. Instead, Java Application Servers can be deployed on a range of platforms that in addition to Unix and NT including HP 3000s, IBM AS/400s and even IBM [or compatible] mainframes. This platform independence eliminates concerns related to scalability and platform obsolescence. Scalability in particular is a major concern when it comes to developing Web applications given the huge pool of potential users on the Internet. Thus, in many instances corporations have to envisage the need to accommodate thousands of concurrent Web users per application. Having the flexibility of being able to easily change platforms, without having to change Application Servers, mitigates scalability concerns since one is not locked into one particular platform technology, operating system [e.g. NT] or vendor.

2. **Effortless Extensibility thanks to Enterprise JavaBeans**: Java, true to contemporary programming philosophy, is inherently object oriented. In addition, there is a widely endorsed standard for server side Java object development and integration, specified by Sun in 1997, known as Enterprise JavaBeans (EJBs). Today most of the major Java Application Servers include comprehensive support for EJBs. With this EJB support the functionality of a Java Application Server can be readily enhanced and extended using different EJB based components from different vendors. Thus, EJBs elevate Java's trademark platform independence to the next logical level – software vendor independence. Given the support for EJBs, one is no longer locked into one particular software vendor when it comes to Java Application Servers. A Java Application Server from one vendor can be enhanced using EJBs from another vendor. It is even possible to mix-and-match EJB components from different Java Application Servers and from different vendors. This is particularly germane when it comes to backend connectivity options to existing systems. A corporation might select a Java Application Server based on its prowess in the areas of directory services, object integration, load-balancing and database access only to discover that this particular product currently does not offer 5250 access to IBM AS/400 systems. Such an omission is no longer a show-stopper. It would, in general, be possible to get the necessary connectivity option, in this example a 5250 access module, in the form of an EJB component from another vendor.

3. **Standardization on Java for Web applications**: Applets written in Java, and dynamically downloaded from a Web server on demand, are by far the most widely used mechanism to extend the functionality of Web pages by transparently adding interactive intelligence at the client side. Today it is difficult to spend any time surfing the Web without encountering a Web page which will automatically download a Java applet to bolster the capability of the Web page or in some cases even the Web Browser being used. Savvy Web designers no longer think of Web pages of consisting entirely of HyperText Markup Language (HTML) based content. Instead they think of Web pages as an amalgamation of HTML and Java applets. Consequently corporations will inevitably end up with in-house Java programming expertise and experience as they start to venture into Web based applications. Furthermore given its Web orientation and platform-independent ubiquity, Java is rapidly becoming the programming language of choice among the software development community. The legions of COBOL programmers that have gamely sustained financial sector applications for the last three decades will over the next decade be replaced by Java programmers. Of that there can be no doubt or debate. Given that Java will invariably play a key role in the development of future Web applications and that corporations will end up with a growing pool of Java aficionados, it makes strategic sense to standardize on Java Application Servers as well. It should also be noted that Java Application Servers do support applications written in Java. Thus, new Web applications as well as applications to integrate existing data sources with Web page content can be written in Java and executed within the context of a Java Application Server.

4. **Architected robustness**: Java is typically an interpreted language which runs within an program execution environment known as a Java Virtual Machine (JVM). JVMs are

architected with comprehensive memory protection and validation to obviate any type of failures due to one program violating the memory space of another component. The interpretive nature of Java coupled with this memory protection ensures that Java applications in general enjoy exceptional 'up-time' and resilience. A Java Application Server, which is but a Java application, should thus prove to be a robust and reliable server-side mission-critical offering capable of delivering the high-availability vital for most financial sector applications.

A Plethora of Clients – Some Thin, Some Sophisticated

Most Web applications are targeted at Web Browser users. Data exchange with these users will be via Web pages – augmented with Java applets where necessary. HTML based Web pages and applets will be downloaded to the Web Browser, from a standard Web Server, using the HyperText Transfer Protocol (HTTP). Security in the form of user authentication and end-to-end data encryption between the Browser and the Server will be realized using either Secure Sockets Layer (SSL) technology or Secure HTTP (HTTPS) which is essentially a form of short-duration, per-transaction SSL.

These applications that rely purely on HTML based Web Pages and Java for their data interchange are known as 'Thin Client' Web applications. The term 'Thin Client' alludes to the fact that the only software required at the client is a basic operating system [e.g. Windows 95] plus a standard Web Browser à la Netscape Navigator or Microsoft's Internet Explorer. No additional software is required to support any dynamically downloaded Java applets since modern Browsers now include a built-in, full-function JVM to execute Java applets. Such Web applications can also thus also be accessed from Network Computers (NCs), in addition to PCs, Apple Macs and Unix workstations.

All Java Application Servers support this type of 'Browser Only' Web applications that rely purely on HTML, HTTP and Java applets for all of their data interchange. The Applications Server, true to the conventions of the Web, will, however, work in tandem with a standard Web Server to download the application specific Web Pages and applets required by the Browser users. Security across the Web will usually be handled using standard SSL or HTTPS between the Browser and the Web Server – in the same manner as if there was no App. Server involved.

Some Web oriented financial applications may, however, require more local processing at the PC/workstation than is feasible with just a 'thin client' approach. In such situations it would normal to have an application program that will execute on the client machine in conjunction with a server component. These client applications, where necessary, will execute alongside a Browser – rather than within the Browser per se as is the case with Java applets invoked through a Web Page. Such client applications could be written in C, Visual Basic, C++ or Java. Java Application Servers will typically support such "sophisticated" [or 'fat'] clients, in addition to, and concurrently, with conventional "thin clients". Thus, corporations can enjoy total flexibility when it comes to the design and implementation of their new 'client-server', Web-centric applications.

The client application could use any Internet Protocol (IP) based mechanism, including Transmission Control Protocol (TCP), User Datagram Protocol or HTTP, to interact with its companion server component executing on the Java Application Server. It would also be possible to use an application specific protocol, on top of IP, or even TCP/IP, to realize the client-server communications. This client-server communication would not occur through a Web Server and would take place directly between the client and the server across the Web or an intranet. Consequently the application, rather than the Java Application Server or a Web Server, will be responsible for any and all security – such as end-to-end data encryption. While Java Application Servers do support client-server applications that rely on this type of non-object oriented communication, it is not the approach they advocate. Java Application Servers promote and encourage object oriented program development.

With Java Application Servers it is possible to use standards-based object invocation and interaction between the client and the server. Today, objects created by different groups or vendors can be uniformly invoked and integrated, irrespective of differences in platform type between the object's source and its destination, using a methodology known as an 'Object Request Broker' (ORB). The industry standard for ORB is the 'Common Object Request Broker Architecture' (CORBA) that was developed by the Object Management Group (OMG). Java Application Servers support CORBA at the server level for server-side applications, as well as CORBA-based object manipulation across the network between clients and a server. The standard for performing CORBA across IP networks such as the Internet and intranets is known as 'Internet Inter-ORB Protocol' (IIOP). Java Application Servers support IIOP for object-based client-server interactions.

The native Java scheme for ORB is known as 'Remote Method Invocation' (RMI). Most Java Application Servers also support RMI. RMI will be used when the client application is an object-oriented application written in Java. A client side Java application differs from a Java applet in that it is not dynamically downloaded from a Web Server as a part of a Web Page related operation. Instead, a Java application is a traditional application that is written in Java. Whereas Java applets execute in the JVM provided by Web Browsers, Java applications run on a JVM provided by the operating system. Thus, Java applications are not tied to Web Browsers or Web Servers. Java applications can, however, run alongside an active Browser.

IIOP and RMI based interactions occur directly between the server component running on a Java Application Server and the relevant clients. These interactions do not go through a Web Server. When using IIOP or RMI, the security measures used between the client and the server will again be the responsibility of the actual application, rather than that of the App. Server or a Web Server.

The types of clients that can be supported with a Java Application Server can thus be summarized as follows:

Client Type	**Client Software**	**Protocol**	**Object Oriented**	**Web Server Involved**	**Client Application Written In**
Thin Client	Web Browser	HTML/Java over HTTP	No	Yes	N/A
Traditional Client	Client Application	Application specific over IP	No	No	C, Visual Basic ...
Object Oriented Client	Object Oriented Client Application	IIOP	Yes	No	C, C++, Visual Basic, Java ...
Java Client	Object-Oriented Java client app.	RMI or IIOP	Yes	No	Java

<u>The Anatomy of a Java Application Server</u>

A typical Java Application Server, as shown in **Fig. 2**, can be thought of as consisting of the following key components:

1. **Integration Engine**: This is the heart of the App. Server and the execution environment in which the Web applications will run. The Integration Engine will be highly object-oriented and support EJBs, CORBA and RMI to facilitate industry standards based object invocation, manipulation and integration. With scalability being a perennial issue vis-à-vis Web applications, nearly all Java Application Servers provide a built-in scheme for load-balancing – whereby multiple instances of the Java Application Server, each on a separate platform, can work together, in parallel, dynamically servicing a large pool of Web users. In some instances,

individual components of the Application Server may be distributed across multiple servers to realize the desired processing and memory requirements. Most offerings will support load-balancing using a least-used, first-available, or random allocation mechanism to distribute new users across the multiple servers. In some instances, the Java Application Server may provide a load-balancing agent, in the form of a 'script', that works directly with a Web Server. With such a scheme, new user requests for the App. Server will automatically be directed to the 'next-in-line' server by the Web Server. The load-balancing mechanism will also provide fault-tolerancy in the form of 'fail-over' protection. Thus, in the event of a server failure, the user 'sessions' being serviced by that server will be transferred over, automatically, to another server.

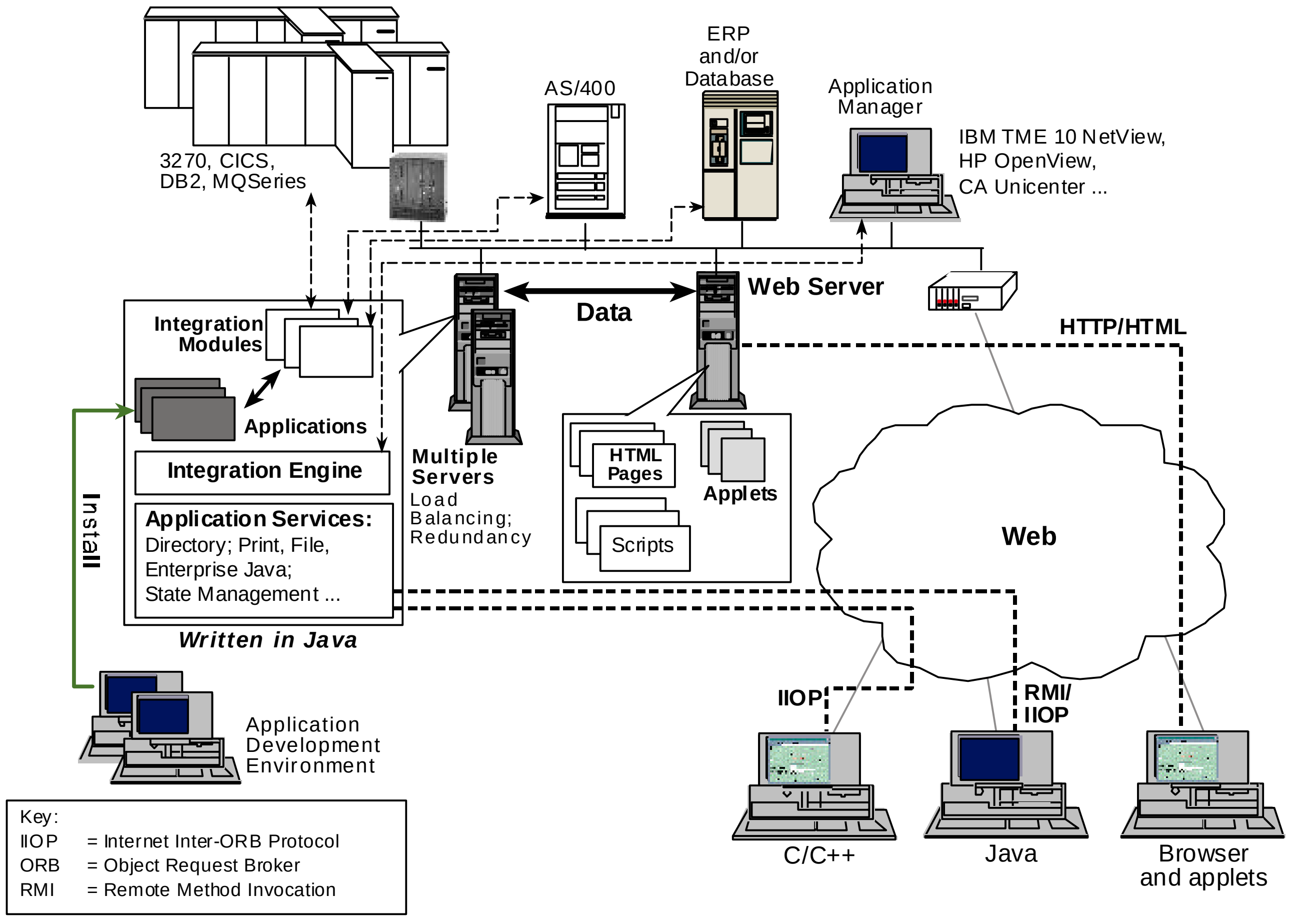

Fig. 2: The generic architecture of Java based Application Servers

2. **Application Services**: These will be a set of core services for the Web applications such as: directory, print, file, time and state management. These days directory services will invariably include support for the de facto industry standard Lightweight Directory Access Protocol (LDAP) and also possibly for the widely used and endorsed Novell Directory Services (NDS). State management helps applications keep track of various events and process these events asynchronously. Some App. Servers will also provide security functions, such as SSL, for use by the Web applications.

3. **Integration Modules or Backend 'Hooks'**: These are the modules that enable the new Web applications to adroitly interact with existing data sources and applications. The common integration modules available with most Java Application Servers include:

- Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) for accessing relational databases
- 3270 and 5250 to enable access to mainframe or AS/400 resident SNA mission-critical applications using terminal emulation
- VT220 type asynchronous terminal mode access to applications running on minicomputers and Unix machines
- Specific interfaces to Enterprise Resource Planning (ERP) applications such as those from SAP, PeopleSoft and BAAN
- CICS interface, typically via the client-server oriented CICS External Call Interface (ECI), to the widely used IBM Transaction Processing subsystem that runs on mainframes, AS/400s, HP Systems, Unix machines and even NT servers
- MQSeries interface to IBM's message queuing based middleware offering that is now being increasingly used to develop new client-server applications in IBM environments
- Specific security server interfaces to enable Web applications to avail themselves of security services - such as user authentication

4. **Application Manager**: This component will typically work in conjunction with a popular Network Management platform such as IBM's TME 10 NetView, HP's OpenView or Computer Associate's Unicenter and enable administrators to monitor and manage the on-going operation of the Web applications, the Integration Modules as well as the Java Application Server itself. Given that the Java Application Server and the Web applications will become key mission-critical resources at most financial institutions, this management capability is crucial to ensuring smooth, trouble-free, high-availability operations.

5. **Application Development Environment**: This component facilitates the development of the Web applications that will run on the Java App. Server. There are, however, two very different schools of thought as to how this key function should be provided vis-à-vis a Java Application Server. Some App. Servers, such as the Bluestone Sapphire/Web, provide an Integrated Development Environment (IDE), replete with development expediting Software Development Kits (SDKs), as a part of the App. Server. Such IDEs, consistent with current program development ideology, tend to be highly visual à la the paradigm popularized by Visual Basic. They support program construction via components that can be 'dragged-and-dropped' on a screen using a mouse. Other Java Application Servers vendors, on the other hand, advocate the notion that there are already plenty of very good Java application development products on the market and that there is no point in trying to reinvent the wheel. BEA Weblogic and Novera's jBusiness fall into this category. Thus, rather than providing their own IDE, they recommend that corporations use any of the popular Java development tools such as Symantec's Visual Café, Inprise's Jbuilder, IBM's VisualAge for Java or Microsoft's InterDev.

Platforms for Java App. Servers

An underlying premise and value proposition of a Java based Application Server is platform independence. In theory one can run a Java App. Server on any system that has a full-function implementation of a JVM. Nonetheless, most Java App. Servers today run on Unix systems and NT Servers. This is mainly because Java App. Server vendors find that these are the most readily accessible boxes for testing and 'certifying' their software.

Customers are also buying into this NT and Unix oriented thinking, at least for the time being, in order to validate and gain experience with this new technology. With Y2K issues still abounding, most do not want to experiment with larger boxes – such as mainframes. Scalability and reliability tends to be the main concern in selecting the appropriate platform for Java App. Servers. Unix systems scale better and are typically more robust than NT Servers but on the other hand tend to be more costly. Given that all App. Servers support load-balancing and fault-tolerance it makes sense in many instances to deploy multiple servers – typically NT Servers.

The beauty of Java App. Servers is that they are not restricted to NT or Unix. Full-function, up to-date JVMs are now available on IBM mainframes [including those running the VM operating system] and AS/400s. A few of the Java App. Server vendors, such as BEA Weblogic, are now promoting the AS/400 as a viable and attractive platform. Some list mainframes as a potential platform in their marketing collateral but do not pursue this claim much further. This has mainly to do with testing and support. Most Java App. Server vendors do not have a background that involves mainframe-based solutions. Neither do they have ready access to mainframes. Thus, testing and supporting their product in a mainframe environment is a real issue. In terms of scalability and reliability, mainframes are in an exceptional class of their own. Financial institutions know this - and to be fair, so do the App. Server vendors. So the current climate will change in the near future, especially once the dust has settled after Y2K. At that point, most likely led by IBM, many vendors will start supporting their Java App. Servers on mainframes.

Bottom Line

Web enabling existing financial applications as well as developing new Web-specific applications will become the number one challenge for most financial institutions as soon as they have successfully overcome their Y2K hurdles. Java Application Servers are an optimum, strategic and cost-effective means for implementing new applications, reusing existing resources, and Web enabling existing applications. Platform-independent Java is rapidly becoming the programming language of choice for the 21st century – and Java Application Servers leverage the power and popularity of Java to facilitate Web integration. Proven and feature-rich Java Application Servers are now available from a variety of vendors. Moreover, the EJB capability of these App. Servers ensure that one can build a customized, best-of-breed Java Application Server using components from different vendors. Financial institutions that are now in the process of evaluating options for Web integration really have no choice bust to seriously consider Java App. Servers as a very strong contender that is likely to meet, if not exceed, all of their current and future requirements relative to the Web.

BIO:

Anura ['SNA'] Gurugé is an Independent Technical Analyst Consultant who specializes in all aspects of contemporary IBM networking. He has first hand, in-depth experience in SNA-Capable i•nets, SNA/APPN/HPR/AnyNet, Frame Relay, Token-Ring switching, ATM, System Management, and *x*DSL technologies. He was actively involved with the Token-Ring switching pioneer Nashoba Networks.

Over the last seven years, he has worked closely with most of the leading bridge/router, intelligent hub, FRAD, Token-Ring Switching and Gateway vendors, and has designed many of the SNA-related features now found on bridge/routers and 'gateways'. He has also helped large IBM customers to re-engineer and totally overhaul their old, SNA-only networks. He is the founder and Chairman of the SNA-Capable i•net Forum [*www.sna-inets.com*].

He is the author of "*Reengineering IBM Networks*" [pp 600; 1996], the best selling "*SNA: Theory and Practice*" [pp 570; 1984], as well as several other books on SNA, APPN and SAA. His latest book "*Integrating TCP/IP i•nets with IBM Data Centers*" will be published in August 1999 by Addison Wesley Longman. He also the Editor of Auerbach's "*Handbook of Communications Systems Management*".

He has published over 260 articles and is the author of the Business Communications Review (BCR) supplements on “*BCR’s Guide to SNA Internetworking*” and “*Beyond SNA Internetworking*”.

He conducts technical and sales training seminars worldwide on a regular basis and has conducted one-day workshops for Networld+InterOp since 1992. He publishes a monthly 16-page Newsletter on Contemporary IBM Internetworking.

In a career spanning 24 years he has held senior technical and marketing roles in IBM, ITT, Northern Telecom, Wang and BBN. He can be contacted at (603) 279-5596 or guruge@cyberportal.net. www.inet-guru.com.