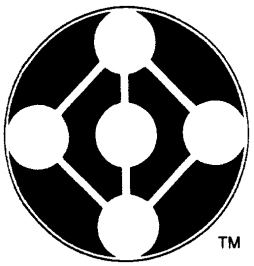


Alternative Technologies

The

Web-Enabled Extraprise

David McGoveran
Alternative Technologies
13150 Highway 9, Suite 123
Boulder Creek, California, 95006
Website: www.AlternativeTech.com
Email: mcgoveran@AlternativeTech.com
Telephone: 831/338-4621 Facsimile: 831/338-3113

Report Number 980805

Disclaimer and Notice

This report is produced and published by Alternative Technologies, Boulder Creek, CA. The information and opinions presented in this report are exclusively those of Alternative Technologies, except where explicitly quoted and referenced. Although reasonable attempts have been made to insure the accuracy of the report, no guarantees or warrantees of correctness are made, either express or implied. Readers are encouraged to verify the opinions stated herein through their own efforts.

Bluestone Software is granted a non-exclusive license for unlimited distribution of this report in its unabridged form. No abridgement of this report may be reproduced in any form or media without the explicit written permission of Alternative Technologies.

For information about this or other reports, or other products and services (including consulting and educational seminars), contact Alternative Technologies directly by telephone, mail, or via our Web site:

Alternative Technologies
13150 Highway 9, Suite 123
Boulder Creek, CA 95006
Telephone: 831/338-4621 • FAX: 831/338-3113
Email: mcgoveran@AlternativeTech.com
Website: www.AlternativeTech.com

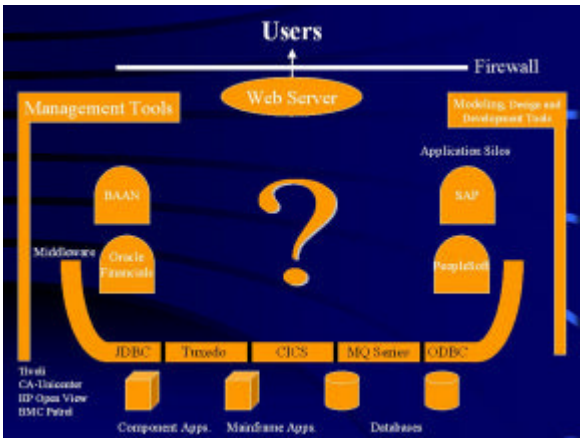
TABLE OF CONTENTS

I. Introduction

On the way to the virtual enterprise[1], existing enterprises must transition from a control-centric approach to an opportunity-driven approach. The world is changing more and more rapidly, forcing businesses to focus on taking advantage of ever new, perhaps transient, opportunities in order to compete. In an opportunity-driven enterprise, importance is placed on agility rather than on control. This transition is being fueled by Web-enablement of business functions. Web-enablement of departmental, cross-departmental, and enterprise business functions increasingly impact the business. With application server technology, a business has an opportunity to absorb that impact while leveraging existing technologies.

But the Web-enabled enterprise is not the end of this progression. The extranet is being used to extend business processes beyond those implemented via the corporate infrastructure. When this is done, the term "enterprise" can no longer describe business activities. In this paper, the term ***extraprise*** is introduced to capture the sense of this developing business style.

As every CIO knows, it is becoming more and more difficult to plan in this world of rapid business change. Such change makes delivering sustainable technology for the extraprise very difficult. In addition, there is a great need to find some way in which to interconnect a wide range of technologies while maintaining control (see Figure 1). Application server technologies have seen renewed interest because of the need to deliver Web-based solutions. However, not all application server technologies can respond to the explosive growth in capacities nor in the high availability requirements that characterize the successful extraprise.



- FIGURE 1 -

This report will help technology managers understand important features of enterprise application server technologies. The focus is on evaluating application server technologies for their ability to support a business as it transitions to an extraprise. In Part II, we discuss the key issues from a product-independent point-of-view. In Part III, we will discuss the relevant

[1] In its most extreme form, the virtual enterprise may have no physical location, no staff of its own, and may reorganize continuously.

features and functions of Bluestone's Sapphire/Web Application Server Framework, which we recently audited.

II. Evaluating Application Server Technologies

Most technology managers are very familiar with the process of selecting a software technology for traditional environments. I.T. departments now have almost two decades of experience with the process of high capacity requirements planning, and with measuring performance and load. Principles developed for the mainframe have been extended to the client/server environment. We understand that a desktop class solution may not address a departmental problem and that a departmental class solution will probably not address an enterprise problem. Even business managers with little technical training understand this classification of the power of software products.

Unfortunately, we have yet to develop an equivalent classification scheme for software in the Web-enabled environment. This can make it difficult for IT to select the appropriate product to meet requirements. There is no concept of a "desktop" application server. Even if deployed as a "departmental" solution, the load on an application server will grow rapidly when it becomes accessible to other departments, the sales force, customers, or the general public. Not all application server products can handle this increased load in an equally graceful manner.

Part of the problem is that traditional measures of load will not suffice for an application server in the extraprise. Numbers of users, transaction rates, database sizes, and so on make sense when applied to traditional applications. But these do not help much in the extraprise. The number of users cannot be used as a successful measure. Depending on the work the user needs to accomplish and their sophistication, the load per user can vary greatly. Even measures such as the number of HTML pages served are not enlightening. Depending on the design, the number of HTML pages served to complete a single business function can vary from one complex (and perhaps poorly designed) page to tens of pages (especially when the user is browsing).

Transactions[2] are the primary unit of load traditionally used to measure performance and scalability in software systems[3]. There are three main types of transaction:

- business transaction — the unit of audit, a business transaction includes the work from one audit point to another audit point

- logical transaction — the unit of consistency, a logical transaction includes the work that will transition the system from one consistent state to another consistent state[4]

---

[2] The term transaction is used here in the formal sense. As such, the load may be read-write (involving data entry or collection) or may be read-only (such as a query load).

[3] Abstract workloads are used in measuring performance on hardware systems, but these are inappropriate units for comparison when measuring application software systems.

[4] A system such as a database is in a consistent state if all known consistency or integrity rules are satisfied.

- physical transaction — the unit of recovery, a physical transaction includes the work that can be restored (recovered) in the event of a system failure

Business transactions are the only type of transaction to which a business manager can relate business performance measures. In general, implementation of a business transaction may require multiple logical transactions and implementation of each logical transaction require multiple physical transactions. From the business perspective, business transactions are the ideal measure of load on a system. Business transactions are the more meaningful type of transaction. By contrast, the variability in number of logical and physical transactions required to accomplish a particular business objective makes these transaction types a relatively poor measure of load.

Although it is a crucial measure of load in a traditional software environment, even business transaction rates may not be the ideal measure of load in a Web-based environment. Here, every request is asynchronous. Also, a single user interaction may trigger multiple business transactions across multiple applications. On the other hand, multiple user interactions may be required for the application server to accumulate all the information necessary for a single business transaction to complete.

A better measure of load in a Web-based environment is the number of interactions, defined as roundtrips between the user and the application server for a controlled exchange of information. In a sense, this approach inverts the traditional view of transactional processing. Emphasis is placed on completing or acknowledging the user request in the same way that we usually emphasize logging the completion of a database transaction[5]. Just as transaction management is essential in traditional enterprise applications, so is interaction management in extraprise applications.[6]

In defining an interaction standard load for performance benchmarking or product comparisons, the relationship to business transactions should be carefully documented.[7] In a typical business application, this relationship cannot be held fixed and will vary. Because the number of interactions depends on so many unpredictable factors, it is essential that the application server be robust in a number of ways. These include scalability, availability, manageability, interoperability, and deployment flexibility. In the next sections, each of these features will be addressed.

---

[5] In this case, we treat the user just as we would any resource manager. Until the user has received completion of the interaction (like the transaction), the interaction is not complete (committed).

[6] Like distributed transactions which involve multiple distributed resource managers (or transaction logs), distributed interactions involving multiple users are also possible. These concepts open the way for a more reliable approach to distributed collaborative computing, which will be essential for some types of extraprise applications. For example, this would be a reasonable way in which to implement a distributed, Web version of real estate escrow process which typically involves interactions among the buyer, seller, agent, escrow officer, and lender.

[7] A preferred approach implements one or more business transactions for every interaction, avoiding the complexities of supporting business transactions that span multiple interactions.

Scalability

Given the need to increase capacity rapidly, it is important that a Web-enabled application server technology does not limit the ability to respond to that need. This requirement implies that the application server architecture be distributed, permit multiple load-balanced copies of any application server, and have no bottlenecks.

In order for the architecture to be distributed, it must be designed and developed as a set of interconnected processes. The function of these processes and how they are interconnected is peculiar to the particular product. If an individual process has multiple combined high level functions, it will almost certainly have multiple load characteristics as well. If these functions share resources, their impact on load will not be strictly additive and the process is very unlikely to scale in a linear fashion. A good design will assign one function per process. It will permit (though not require) each process to be associated with its own physical resources or platform. This association is called ***affinity***.

A special form of affinity is called ***session affinity***. There are two important cases when session affinity is advantageous. First, an application server might be designed to maintain state (as might be appropriate for security or performance reasons, or because the application is designed to be procedural). Second, it may require session-specific allocation of resources. In both cases, state must be re-synchronized across application server instances and resources need to be reallocated if instances change for the session. Session affinity guarantees that the client is returned to the same application server process or instance. It eliminates the overhead that would occurs if instances change, and improves scalability.

Few designs provide application server instances that can support an open-ended load. If too much load is assigned to a particular instance, that server instance will become a bottleneck. As a result, it is sometimes desirable to increase the number of instances assigned to a given function while allocating additional physical resources to those instances. However, even if instances for a particular function are assigned to multiple platforms, assigning too much load to the instances on a single platform will cause that platform to become a bottleneck. The solution to this dilemma is automatic and ***dynamic real-time load balancing*** across all resources.

In an application, the context necessary to support completion of a request and its associated response, or between a sequence of requests, is called the ***state***. It must be maintained and correctly transitioned if there is to be any consistency in the application. The process by which state is coordinated among the components of an application is called ***state synchronization***. The greater the need for state synchronization among an application's components, the more monolithic the behavior of the application becomes. Often some components must wait for another component in order to synchronize state. The latter component then dictates the throughput of the system, becoming a bottleneck if it is not capable of handling the load.

The time cost incurred when an application component must wait in order to synchronize state depends on two factors: (1) the performance of the component with which it is synchronizing state and (2) the communications mechanism being used. If state synchronization occurs in

shared memory, the cost is relatively low. By contrast, if it occurs across platforms, the cost is relatively high. Ideally, of course, state would never have to be maintained or synchronized. Such applications are said to be ***stateless***. In principle, all applications could be stateless. In practice few applications are designed to be.

Given the time cost and need to avoid monolithic behavior, state management issues are especially important in a distributed application. For example, in a traditional client/server environment, both the client and the server maintain a portion of any necessary state. This is possible because the connection is usually synchronous, meaning that the client portion does not proceed until it has received a response to its most recent request. Of course, if the client process fails, state is lost and at least the most recent request must be resubmitted during recovery. On the other hand, if excessive state synchronization with the server is required, the time cost across the network is significant and the server can become a bottleneck. Failure to minimize the need for state synchronization and manage it properly are among the most common reasons for loss of scalability among failed client/server applications.

In a Web-enabled environment, the connection is asynchronous, permitting the client to proceed with other requests prior to receiving a response. This means that the entire state or context of the communication must be maintained in some location, unless the application is designed to be stateless. There are five basic approaches to maintaining state in a Web-based environment. These include maintenance in the browser client, in the Web server, in the application server, in middleware, or in a persistent state server.

One method is to maintain state in the Web client as a "cookie" or in HTML. While this method is scalable, it places reliance on the most uncontrolled portion of the architecture. Loss of the client (due to, for example, unexpected loss of connection or platform failure) can have a serious impact on application and database consistency. Because an interaction may generate multiple transactions across application servers, coordinating the application recovery in the face of Web client failure becomes extremely complex and error prone. In particular, it is possible that only some of the transactions are completed without the client-held state having been re-synchronized.

Maintenance in the Web server is a second method, but this method quickly becomes a bottleneck under heavy load. As a result, the application should require maintenance only of "light" state information for each client, thereby minimizing the incremental load. A third method is to maintain state in the application server. This method removes the risk that state will be lost with respect to the application server context, and consequentially that consistency will be jeopardized. However, the application server can easily become a bottleneck because the load due to state maintenance (and recovery) can be quite high compared to execution of application functionality.[8] Likewise, state can be maintained in middleware, but this fourth method also easily becomes a bottleneck and is difficult to recover when it becomes overloaded. The fifth

[8] Even though the time cost of state synchronization is low if done in shared memory (as it might well be when state is maintained in the application server), the approach still becomes unviable. It places availability and durability in excessive jeopardy and sessions block waiting for access to shared memory during state synchronization.

method is the use of a persistent ***state server***. By assigning state maintenance functionality to its own server process, scalability is assured and the potential for bottleneck is greatly reduced over other methods.

Another important aspect of scalability is the load that a particular application server instance can support. This is affected by whether or not the application server is multi-threaded. Until recently, most application servers were not multi-threaded. The difficulties of developing thread-safe, multi-threaded software have been greatly reduced by the recent introduction of Java. Because Java is an inherently multi-threaded, application servers that are 100% Java are more likely to provide the throughput benefits of multi-threading.

Availability

Availability is a familiar requirement of enterprise applications. Expected loads are fairly predictable and permit considerable capacity planning. In such environments, TP monitors, highly available DBMSs, hardware redundancy, and highly available operating systems with known failure rates and planned maintenance cycles are used to meet specific availability requirements. Like enterprise applications, a Web-enabled application environment for the extraprise cannot afford significant downtime. Extraprise application servers should provide support for TP monitors and highly available DBMSs. There should be no restrictions on platform support that would preclude the use of hardware redundancy or fault tolerant operating systems.

While traditional methods to improve availability are still required, the unpredictability of loads in an extraprise application brings new concerns. In particular, the interaction must be completed even if the client, application server, or platform fails. When a highly available state server is a part of the architecture, ***service migration*** techniques can be an effective response to a failure. This technique requires either a multi-instance application server or the ability to start an application server instance on a different platform. When a failure occurs, the request processing is restarted automatically on a different application server. Service migration is a familiar feature of TP monitors. However, because of the asynchronous nature of the Web-environment, service migration is difficult to implement between Web-client and application servers using a TP monitor. The corresponding functionality should be provided by a Web-enabled application server environment.

Application isolation is a deployment style which, among other things, can improve availability. In this style, applications can be deployed as self-contained application servers. This enables service migration at the application level, as well as enhancing the impact of load balancing techniques on availability. It also permits the application resource usage, performance, and availability to be monitored, and resources to be allocated to the application as needed.

Manageability

The extraprise application environment can become extremely complex. It can involve many application servers, browsers, databases, distributed components, and distributed applications

(including component-based, desktop, and legacy). Integration with a management tool is required if such system complexity is to be monitored and controlled. Ideally, the application server environment will have a native management tool. Ease of use of such a tool is essential in the complex environments that are needed to support the extraprise. A graphical console for both monitoring and control functions will make it possible to visualize the environment quickly and easily.

Because the extraprise application is inherently distributed, integration with management tools should be agent-based. An agent-based approach permits real-time monitoring (including event notification and alarms) and the ability to detect availability status even if the application service is shut down or has been moved to a different platform. It can also provide run-time management including a means to start, stop, and check the health of any platform or component of the application. Real-time monitoring should also support the collection of statistics on usage, loads, response times, and so on. Once collected, it should be possible to analyze them and produce trend reports periodically. Utilization and service levels are particularly important for the extraprise.

With agent-based management, integration with existing, third-party system management utilities is possible. At the most basic level, this integration should provide simple status monitoring. More advanced monitoring should provide statistics collection about the environment. Tighter integration with third-party system management utilities should permit configuration information to be passed to the application server management facility, thereby providing a single point of management and control across the enterprise.

Interoperability

An extraprise application server must be able to integrate with a wide variety of packaged application software, component based applications, legacy applications, and middleware. Integration is usually based on a relatively small amount of special software. This software goes by a variety of names including adapter, bridge, connector, driver, and integration module. For purposes of this report, we will use the term ***integration module***.[9]

There are many types of integration modules, which differ in the degree of software integration supported. Data integration modules provide data transfer to and from an application, but do not address either process or transactional concerns. A common variant of this integration module type provides data integration through file-based import and export capability. File-based (or file format) integration modules can be used to rapidly replace manual or semi-manual pre-existing application integration solutions that are file-based or which use EDI. Data integration modules are relatively easy to develop and deploy, but are not well-suited to online electronic commerce applications. They are best suited to environments that can tolerant either batch update or in which update transactions involve no response.

[9] As we will see, this is the term used by Bluestone. While uncommon in the industry, it emphasizes the need for integration in extraprise applications.

API-based integration modules use an application-specific API. In most cases, the API provides a means to submit business transactions to the application. In some cases, a lower level API is provided and the user must determine how to identify transaction boundaries. In other cases, the API is used to develop messages that provide input to an application's screens, panels, or pages. API-based integration modules are best suited to packaged applications.

Middleware integration modules provide access to middleware facilities. Middleware[10] such as CICS, Tuxedo, MQSeries, MSMQ, COM services, CORBA services, ODBC (for connectivity to DBMSs), email (SMTP), security services, Web servers, browsers, and other facilities are used to provide message and transaction support. These are often the required vehicle for integration with legacy applications, component-based applications, and existing application silos. When middleware integration modules are used for component-based applications, support for either COM or CORBA services is essential.

Component integration modules are relatively new in the market. These integration modules provide easy integration with Java applets and Servlets, Java Beans, Enterprise Java Beans, CORBA objects, ActiveX components, and others. An important feature of component integration modules is ***dynamic introspection***, which permits access to the interfaces of broker-registered components. Component integration modules are best suited to component based environments or, in conjunction with middleware integration modules, integration of components and legacy applications.

Custom integration modules can be of any type. Development of an integration module usually requires a developer kit and may be supported by an integrated developer's environment as well. Like component integration modules, custom integration modules may support dynamic introspection. The languages supported are also important: Java and C++ are most common, but a wide variety of other languages are desirable.

Deployment Flexibility

It is difficult to predict how an extraprise application will need to be deployed. Architectural or other limitations on the method of deployment are therefore undesireable. Thin client Web-based applications are likely to be a significant part of the deployed architecture. However, this does not preclude the need to support fat clients, which may be appropriate (for example, when the client must perform heavy analytical work on data extracts). Similarly, peer-to-peer and multi-tier client/server deployment architectures may be required for portions of the extraprise application.

The application server should be platform and database independent. This means that standards must be supported. It also suggests that Java deployment capability would be advantageous. In the event that Java is not available or is considered to immature, C/C++ deployment is still a

[10] As used here, middleware refers to products and services that support application-independent services such as messaging and transactions. Application servers are also middleware, but generally provide a specific business related service.

good portability alternative. Changing the deployment language should not require redevelopment.

Most deployment choices are made in or enabled by the development environment. An integrated development environment should be available. It should support the development of applications with any user interface technology and developed in any language, as these choices greatly affect the deployment. Just like enterprise class applications, the development environment should support features such as source code control, project management, team development, and the ability to merge multiple projects. These features affect the ability to control and manage deployment choices, in addition to the obvious development value.

Even though deployment options may have been determined in the development environment, the actual physical deployment should be dynamic. This feature permits an application server to be moved from one platform to another even while the application is running. This might be done, for example, because the initial platform requires maintenance or because the new deployment provides a better allocation of resources.

III. The Sapphire/Web Application Server Framework

Bluestone Software's Sapphire/Web Application Server Framework is an example of today's application server technology. The product has a number of components which are used to address the issues of scalability, availability, manageability, and deployment flexibility. The components are:

- Sapphire/Universal Business Server
- Sapphire/Application Manager
- Sapphire/Enterprise Deployment Kit
- Sapphire/Integration Modules
- Sapphire/Developer

Scalability

The Sapphire/Universal Business Server is Bluestone's highly scalable application server deployment architecture. It is a multi-process environment and a Sapphire/Web application server is multi-threaded. It's environment includes a persistent State Server and a Load Balance Broker.

The Load Balance Broker component provides automatic and dynamic real-time load balancing. This component runs in the Web server as an extension (ISAPI, NSAPI, or CGI), as a servlet, or inside the firewall as a proxy. It can also run across servers, thereby supporting very large sites that distribute load across multiple Web servers. The load is evaluated based on the number of active requests being processed by an application server (and application server availability). This facility routes incoming requests, starts additional application servers as needed, and helps utilize CPU resources efficiently. Not only is it possible to distribute the load across multiple

platforms, the load can also be distributed across multiple instances. These instances may either be replicas of a single application, single function, or may be a set of related functionality implemented as distributed processes. As loads vary, the resources allocated to an application server instance are varied as well. Within any particular instance, the number of threads assigned to a task may be varied.

A high-volume persistent State Server is provided for state management, although client, Web server, application, and zero state management options are also supported. State can be cached to improve performance. Global, application, session, user, object, and string states can be optionally managed. When state is maintained in the application server or the persistent State Server, the Load Balance Broker can provide *session affinity*. Session affinity means that a client session will be consistently directed to the same application server instance. This allows the application server instance to use non-persistent cached objects across interactions. Session affinity can be extremely important for high performance, secure applications such as *electronic commerce transactions*. The feature would also be important, for example, if establishing a database connection for each request would impair performance.[11] In this case, the database connection could be cached in application server memory by Sapphire/Web's Live Object Cache. Objects stored in the Live Object Cache can be set to refresh automatically if referenced after a selected "time out." To enhance throughput of an application server, persistent data can be stored in memory on a per-session basis.

Availability

Sapphire/Web applications can be deployed in highly available and fault tolerant environments without restriction. In addition to this deployment flexibility, the Sapphire/Universal Business Server and the agent-based Sapphire/Application Manager (see below) use a number of techniques to improve availability.

In a distributed application, failures can occur at many levels. These include threads, processes, server platforms, network connections, and clients. If the Load Balance Broker or a Sapphire/Application Manager agent detects a failure, a new instance can be started automatically in response to an alarm set through the Sapphire/Application Manager console. This provides failover, for example, across application server instances: any further browser interactions will be routed automatically to a live instance. Sapphire/Web supports the *application isolation* deployment style, which can be used to enhance recovery.

An application server instance is automatically stopped if it receives no requests within a settable time-out, thereby preventing it from consuming resources unnecessarily. Servers can be added or removed while the system is online. Simultaneously, an alternate resource is assigned to the outstanding task automatically by the Load Balance Broker component. Because applications can be run on any platform in the network, the ability to route requests to any available resource is possible. The Load Balance Broker supports service migration.

---

[11] Note that this is a somewhat artificial example, selected for purposes of familiarity. Which objects are stored in the Live Object Cache is at the discretion of the developer. Sapphire/Web actually provides other efficient mechanisms for connection caching.

Manageability

The Sapphire/Application Manager is an agent-based sub-system for monitoring and controlling the entire Sapphire Application Server environment, from the Web client to the application server. It consists of agents, a fault-tolerant rules-based server engine, application mangers, and one or more GUI consoles. Agents communicate between application servers, state servers, data servers, and other Sapphire/Application Manager components. The GUI console can control multiple application managers and an application manager can accept commands from multiple consoles. The console is used to configure load balancing rules, change application server platforms, and control the number of running application server instances. All of this can be done online and in real-time. Application server deployment options (set prior to deployment) let administrators set the number of allowable requests per application server instance, the maximum number of cached database connections, and maximum application server idle time.

Agents reside on each platform, and can be assigned to a platform, Web server, database server, or an individual application server. They collect statistics, control application servers, and monitor applications. Statistics are logged by agents to a database, from which trend and service level reports can be generated. This also permits chargeback accounting. Events can be defined and named, although they cannot be grouped. Transactions can have a maximum elapsed time assigned, whereupon an event is logged. Alarms and alerts are supported. For example, an alarm is generated if performance of an agent-monitored component drops below specified parameters. In addition, responses (such as starting a new instance, sending email, or a pager alert) to an alarm can be automated.

Tight integration is provided with CA Unicenter and is available for IBM's Tivoli. In fact, any SNMP compatible system management software (such as HP OpenView and BMC Patrol) can be integrated with the Sapphire/Application Manager.

Interoperability

Sapphire/Enterprise Deployment Kit

The Sapphire/Enterprise Deployment Kit can be used to develop both *custom* and *component integration modules*. The Bluestone name for an integration module is a SIM (Sapphire/Integration Module). These integration modules can interoperate with COM, CORBA, MTS, JavaBeans, and Enterprise JavaBeans. Object classes are auto-detected and dynamic introspection of Visibroker, Orbix, RMI, and COM/DCOM objects is supported via the Java Object Binder component. ***Dynamic introspection*** allows all object methods and properties to be displayed so that they can be the input and output parameters/methods for the custom integration module wrapper. The custom integration module wrapper implements a developer-defined set of functionality, which can be the entire component function or some subset of behavior.

Sapphire/Integration Modules

In addition to custom integration modules, Bluestone offers a number of pre-built integration modules. These include:

- *API-based integration modules* for SAP and PeopleSoft.
- *Middleware integration modules* for Tuxedo, MQSeries, CICS, Open Market, LDAP, email, ODBC/JDBC, and others.
- A special class of integration modules support modeling. Integration modules for Logicworks ERWIN and Rational's UML are available.

Additional pre-built integration modules are being developed by Bluestone and third party software vendors.

Deployment Flexibility

Sapphire/Developer

As noted earlier, a product's development environment often determines the possible deployment options. Sapphire/Developer is an Integrated Development Environment which enables many deployment options. Sapphire/Developer components include wizards, project management tools, object management tools, a structured editor for Java or C++, database tools, tag editor, and an extensive object library. Users can plug in their favorite Java GUI development tool, tag editor, modeling, Java or Web testing tool, or source code control systems as well. These facilities work together to provide broad deployment flexibility and make it easier to use the expertise of existing development teams.

Applications developed with Sapphire/Developer can use (and integrate) multiple user interface technologies including HTML, DHTML, HDML, XML, Java, Java Script, ActiveX, VB Script, VRML, and others. Application isolation is supported: An application can be run on an application server or can be wrapped as an application server itself. Developed applications are double byte[12] and can be fully localized, permitting the deployment of global applications.

Any Java Virtual Machine (regardless of platform) can be used to deploy Java-based Sapphire/Web applications. Java clients can be deployed via IIOP or RMI, bypassing the Web server. If IIOP is not available, automatic failover to HTTP helps insure access. Because 100% Pure Java is used in Sapphire/Web Java implementations, RMI methods can be called directly from either the server or the client. In addition, C/C++ based Sapphire/Web applications can be deployed on Windows 95, Windows NT, DEC Alpha NT, SunOS, Solaris, IBM AIX, HP-UX, SGI IRIX, and DEC Alpha UNIX. IT departments will find few restrictions with respect to the deployment environment.

[12] Sapphire/Developer itself was not double-byte at the time of this audit.

The Sapphire/Web deployment style is important in managing risk during application development and deployment, or during operation due to failures. Its application isolation capabilities permit each application or application component to be separately maintained, tested, deployed, and scaled, while being managed and configured as a single entity. This prevents individual components from becoming a bottleneck, or from crashing the entire Web site or application on a failure. Application components can be deployed as Java applications, Servlets, JavaBeans, Enterprise Java Beans (EJB), CORBA service, NSAPI, ISAPI, WAI, Oracle's WRB-API, Active Server Pages, or Fast CGI. Any JavaBean can be wrapped as an ActiveX component. These facilities enable existing IT investments to be preserved by providing component-level and cross-platform interoperation.

Middleware deployment options include IIOP, RMI, or DCOM, as well as various ORBs. Licenses for Visigenic's Visibroker and RMI are bundled. Iona's ORBIX and Sun's JavaIDL are supported.

Multiple projects can be managed and isolated, projects can be merged, projects can be reused, and portions of projects can be reused. Single level inheritance is supported. Name conflicts are automatically resolved.

Multiple DBMSs are supported using native access including Oracle, Sybase, MS SQL Server, MS Access, Informix, and DB2. Any ODBC or JDBC accessible DBMS can be used as well. Microsoft's ADO can be used to access Microsoft SQL Server and other Microsoft DBMSs such as Microsoft Access.

A variety of mechanisms can be used in deploying secure Sapphire/Web applications. SSL, SHTTP, custom access control lists, certificates, and third party solutions such as Open Market (for secure e-commerce) and Gradient (for sophisticated access control lists) are supported. Login authorization can be based on operating system, LDAP, DBMS, or any combination. Third-party encryption services (such as RSA, or any private or public key encryption technique) are supported, and a lightweight encryption routine is included.

In addition to the deployment options available for applications developed with Sapphire/Developer, Sapphire/Developer can run on a variety of hardware platforms and operating systems. These include Windows 95, Windows NT, DEC Alpha NT, SunOS, Solaris, IBM AIX, HP-UX, SGI IRIX, and DEC Alpha UNIX. This fact means that developers can work in a familiar environment.

Sapphire/Application Manager

Sapphire/Application Manager provides dynamic physical deployment. An application server instance can be moved from one platform to another using a simple drag-and-drop metaphor. By monitoring performance of an application server, Sapphire/Application Manager can be used to automatically deploy another instance on a designated platform if performance drops below preset standards. A number of configuration parameters important to efficient load balancing

(for example, cached connections, number of active users allowed on a server, and max idle time) can be dynamically redefined.

IV. Conclusions

As businesses attempt to attain the agility demanded of an opportunity-driven approach, Web-enabled application servers will be a key enabling technology. While there are many Web-enabled application server products, not all of them provide the degree of support for scalability, availability, manageability, interoperability, and deployment flexibility that will be demanded by the extraprise. Some products have yet to provide the transactional support demanded by the enterprise. Other products attempt to replicate a transactional focus, but do not give sufficient attention to the interaction. Neither group of products will meet the needs of the extraprise.

Sapphire/Web is uniquely suited to the requirements imposed by the enterprise and the extraprise. Its focus on Enterprise Interaction Management addresses the crucial measure of load which we identified early in this report. Although there are many Web development and application server environments, few have breadth and depth of support necessary for high-end applications. Requirements such as scalability, availability, manageability, interoperability, and deployment flexibility are well-addressed by the components of the Sapphire/Web Application Server Framework. In review, the contribution of these components is as follows:

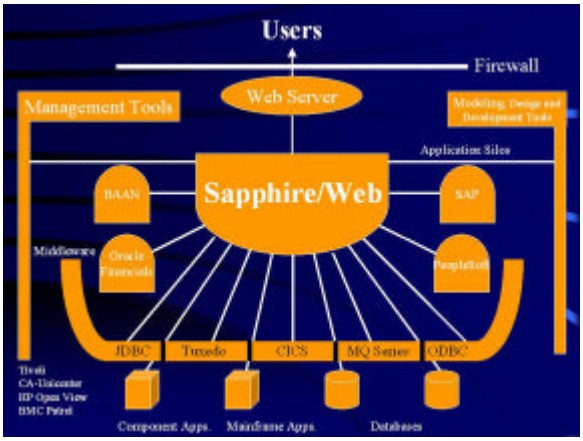
- Sapphire/Universal Business Server — This component is designed to support both highly scalable and available, fault tolerant applications. Its architectural features avoid common bottlenecks and its facilities leave open the ability to meet increasing demands as businesses move from the enterprise to the extraprise.

- Sapphire/Application Manager — This component provides the robust single point of control system management that is essential in a Web-based environment. Its interoperability with third-party system management tools through standards permit integration with existing applications environments. It also enhances availability.

- Sapphire/Enterprise Deployment Kit and Sapphire/Integration Modules — Interoperability with a wide range of existing applications, new applications, middleware, and applications architectures through integration modules guarantees the ability to leverage technology investments. Support for additional pre-built integration modules is desirable, especially for packaged application software and can be expected in future releases.

- Sapphire/Developer — The Sapphire/Web integrated development environment is designed for serious, high-end application development. Wizards and drag-and-drop tools make it relatively easy for the developer to address issues of scalabilty, availability, and interoperability, although these issues imply a degree of complexity that is not likely to be familiar to HTML developers. Sapphire/Developer supports the wide range of environments and technologies (for both development and deployment) that is essential in the complex world of the enterprise and the extraprise.

In conclusion, Sapphire/Web may well be the ideal way in which to interconnect the wide range of technologies found in today's businesses (see Figure 2). IT organizations facing the pressures of electronic commerce, supply chain integration, and similar rapid changes need the capabilities supplied by Sapphire/Web. Businesses transitioning to an extraprise, and facing the inherent business and technological difficulties, will find Sapphire/Web an invaluable product.



- FIGURE 2 -

About Alternative Technologies and David McGoveran

Alternative Technologies was founded in 1976 by David McGoveran, who also serves as the company's president. The company provides a range of industry analyst, business and technical consulting, and educational services. It is widely known for its pioneering contributions to distributed applications, object oriented programming, and relational database technology. For additional information, Alternative Technologies' Website URL is www.AlternativeTech.com.