Under the J2EE Umbrella

Bob Brown has approximately 15 years' experience as a software researcher and developer, and in tertiary-level lecturing throughout the world. Brown is a regular presenter at conferences and events and is co-author of the Prentice-Hall book ***JAVA Thin-Client Programming for a Network Computing Environment.***
E-mail: bob@transentia.com.au

Brisbane-based Transentia Pty Ltd provides specialist consulting, development and training services in such technologies as J2EE, Java, Linux, CORBA and XML.
Visit: http://www.transentia.com.au

J2EE introduces a variety of key technologies aimed at making development of high-end systems easier, says Bob Brown

Introduction

Sun Microsystems' Java 2 Enterprise Edition (J2EE) is a specification for an open, Object-based middleware platform that is increasingly finding favour and application as a tool for bringing advanced functionality to Web-accessible systems.

Background

From the Web's earliest days a progression of technologies ranging from static HTML through HTML+CGI scripts to Java-based Servlets and Java Server Pages has emerged. The desire to use simple resources such as hit counters and direct two-tier access to stored data has also evolved into the need to provide scalable multi-tier access to complex database-based applications and sophisticated enterprise-level infrastructure such as SAP/PeopleSoft, etc.

The J2EE arose from Sun's realisation that Internet-accessible systems are increasingly being called upon to provide ever more complex functionality in ever-shortening timeframes.

While cherry-picking much that was good from various earlier distributed systems development platforms (notably CORBA), the J2EE represents a simplified (but nonetheless extremely powerful) technology that is augmented with a smattering of modern object-based ideas.

Tool support for building network-based systems has always been woefully lacking, making it very hard to design and model complex systems. To address this, Sun is attempting to make J2EE specify both a powerful execution environment for enterprise applications as well as a framework within which it is possible to easily create solutions while relying on powerful tool support. A number of vendors (including Rational, TogetherSoft and WebGain) are stepping up to the challenge and creating high-end (typically UML-oriented) design tools that make it possible to design, develop, deploy and test complex J2EE-based systems with minimal coding.

A related issue is that Sun wants to foster a server-based, enterprise-level component marketplace (in much the same way that Microsoft has for ActiveX and is now developing alongside its .NET platform). As J2EE advances, we should see the development and introduction of enterprise-level 'shrink-wrapped' software.

A final major driver underlying the J2EE's development is the need for software to be developed according to 'Internet Speed'. The strong competitive environment that existed prior to the Internet crash mandated the rapid development of correct highly functional software.

Availability

Most of today's major vendors provide J2EE-compliant application server products, including Sun (naturally), BEA, IBM, Oracle, HP and Borland. A number of less well-known players such as Pramati, Orion and Lutris also offer quality products perhaps equally capable in many respects but typically at a much lower cost than for those available from the first-tier vendors. The Open-Source community is also well represented with the extremely popular JBoss application server which, although not an officially certified J2EE platform, is still widely used and very well thought of in the J2EE community.

This wide spectrum of high-quality implementations is one of J2EE's great strengths: using an Open-Source tool like JBoss it is possible to start developing very sophisticated projects for very little

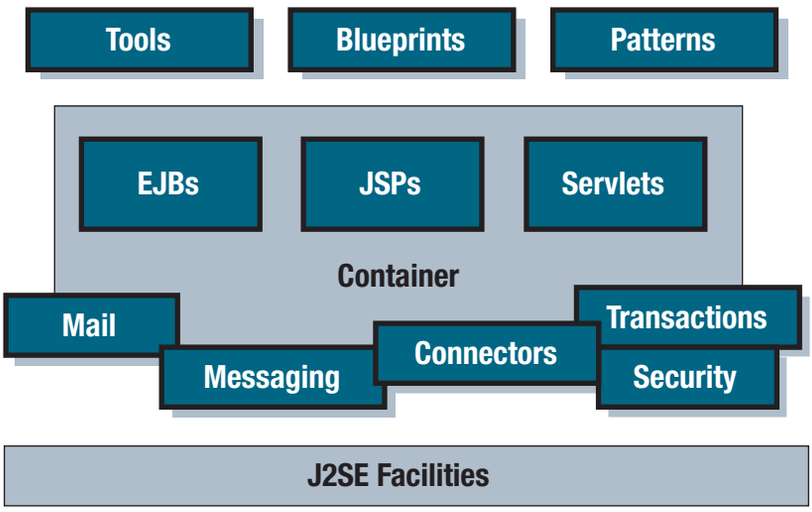

Figure 1 **J2EE major components**

capital outlay. As the project moves to final deployment or as the developed system needs to scale, it is a relatively easy task to move 'up the food chain' to a more powerful and complex platform purchased from a higher-tier vendor.

Architecture

Currently in version 1.3.1, the J2EE platform provides a very extensive framework. As shown in Figure 1, the J2EE incorporates both old and new technologies.

Since the J2EE is based on the Java 2 Standard Edition (J2SE), a large proportion of it is familiar to existing Java developers.

Incorporated into the J2EE from the J2SE are:

- **Applets –** provide for extensible functionality in a browser
- **JavaBeans –** a tool-oriented technique for building component-based applications and frameworks
- **Java Database Connectivity (JDBC) –** facilitates the use of most SQL-addressable data sources
- **Java Naming and Directory Interface (JNDI) –** provides a standardised interface to the multiplicity of directory services typically found scattered throughout an enterprise
- **Java Remote Method Invocation (RMI) –** allows one object to invoke methods on a second object that is located in a separate Java Virtual Machine. The version of RMI now specified by the J2EE makes use of the mature Internet Inter-Orb Protocol standard (IIOP) and so allows Java to interact with systems written in a variety of other languages: C/C++, SmallTalk, COBOL, Ada, etc.

Other familiar technologies that have been gathered under the J2EE umbrella and subsequently augmented include:

- **Java Servlets –** a very common technology that provides an execution and extensibility mechanism for the server
- **Java Server Pages (JSP) –** originally a Java adaptation of Microsoft's Active Server Pages. In effect, a JSP is an 'inverted' Servlet: where a Servlet is a Java class that contains processing logic and may produce HTML, a JSP is an HTML page that contains embedded Java code. JSPs provide a presentation/template system and tool-oriented interface to Java Servlets. They also help solve the problem where graphic designers (who are interested in the HTML markup and look-and-feel of a project) and developers (who are concerned with the logic of an application) end up tripping over each others' feet while working on the same section of a project, as they tend to do when Servlets (which inter-mix markup and logic within one

A final major driver underlying J2EE's development is the need for software to be developed according to 'Internet speed'

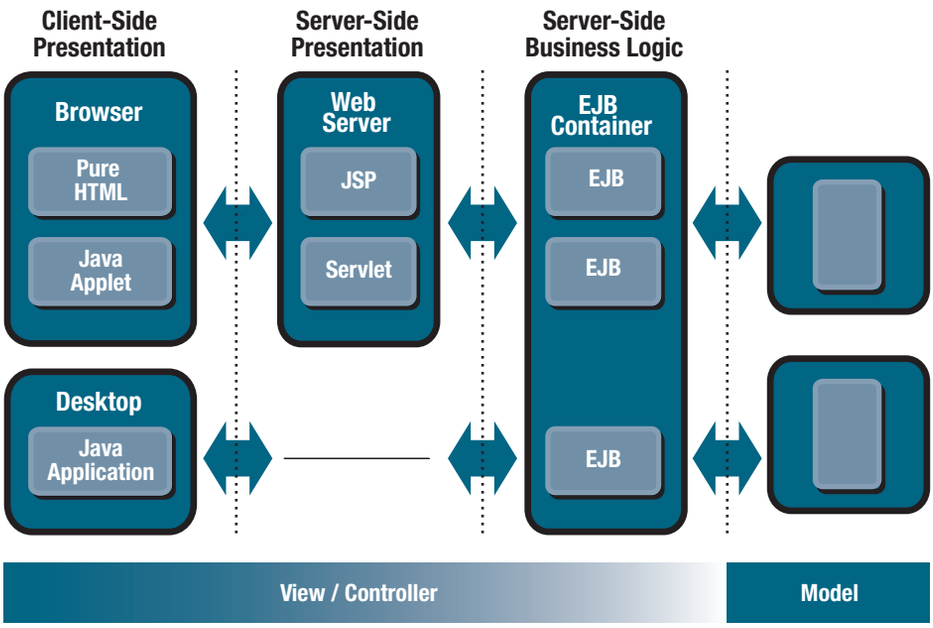

Figure 2 **Model-View-Controller (MVC) in J2EE context**

application) are used for dynamically generating HTML pages.

Java developers interested in interacting with legacy Message Oriented Middleware systems, or who have an interest in supporting asynchronous or disconnected modes of operation will also be relieved to know that J2EE incorporates the pre-existing Java Messaging Service (JMS) and JavaMail technologies.

In addition to gathering together a number of pre-existing elements, the J2EE introduces a variety of key technologies aimed at making development of high-end systems easier. Some of the highlights include:

- **Java Transaction API (JTA), and Java Transaction Service (JTS) -** JTA provides a standardised Application Programmer Interface for transaction handling and JTS effectively provides a Java implementation of JTA that supports distributed transactions.
- **Enterprise JavaBeans (EJB) –** a component architecture for the construction of enterprise-level services. More on this later.
- **Custom tags (usually called 'taglibs') –** provide a simple way to build reusable code blocks, and allow for the clear separation of processing and presentation within a JSP project. Taglibs are very valuable in mid-to-large projects and also for those projects where the look and feel of a page is in the hands of a designer whereas the provision of business functionality is the remit of a separate developer. As with much of the more recent J2EE activity, custom tags also provide a foundation for tool developers and ease the creation of reusable code.
- **Java Connector Architecture (JCA) –** sits beneath the J2EE platform and facilitates the integration of various high-end information systems, such as SAP, CICS and Oracle Financials, allowing them to be plugged into any J2EE environment and accessed in a standardised fashion as 'resources'

EJBs are targeted at developers who wish to be able to design and build complex Object-based systems and also at tool suppliers who need clear, standard APIs with which to work. EJBs are thus more suited to the needs of the enterprise than Java Servlets, which essentially only provide a simple execution mechanism.

J2EE defines three basic types of EJB:

- **Entity –** allows for the definition of persistent storage services and provides an Object-Relational mapping facility
- **Session –** represents a client to the application. The J2EE defines two types of session bean: stateless session beans often represent a single-step workflow or provide some general-purpose functionality for multiple clients, whereas stateful session beans maintain a sequence of interactions with a single client over time.
- **Message-Driven –** provides for asynchronous invocation of a service

Figure 2 (above) shows the unifying idea behind the J2EE: that of Model-View-Controller (MVC). System designs may be decomposed into Models (sources of data such as a database or file; in the J2EE world, this is typically modelled as Entity

Now is the time to investigate whether J2EE is an appropriate tool for tackling your organisation's needs

EJB), Views (a presentation of the current state of a model; represented in the J2EE by JSPs/Servlets/ custom tags) and Controllers (manipulators of the model; typically Servlets, Session EJB or Message-Driven EJB).

The simplicity of MVC means that it is easy to design and build for, but at the same time it maintains a great deal of flexibility and makes it possible to evolve a system in the face of new requirements and changing environments.

A key aspect to the J2EE is control by interposition: all server-side code 'lives' within a container, which controls all aspects of an EJB or JSP/Servlet's existence. It does this by placing itself between all incoming requests from clients and also between all interactions with external resources, be they databases, directory services, or other resources.

No code that is developer-written is ever accessed directly or ever accesses resources directly: as Figure 3 shows, the container mediates everything. (Of course, to do its job, the container may request resources and services from the underlying application server proper.)

The container provides a single point of control for such system aspects as transactions, security, concurrency, persistence, relationships and, in some cases, clustering and failover as well. The API between the container and the code it wraps is small and well defined, making it simple to work with and (returning to the common theme) amenable to tool utilisation. The container also imposes a well-defined lifecycle on wrapped code that further eases the demand on the programmer and simultaneously eases life for our putative J2EE-aware tool.

While retaining simplicity, the J2EE does not throw away power. The operation of the container is easily configured in a declarative fashion by means of an XML-based deployment descriptor. This provides configuration information guiding the container's operation. Among other things, it is possible to specify which individual methods can be invoked and by whom, which methods should be infected with a transaction and which should establish a new transactional boundary, which database table fields are mapped to particular EJB properties and which of these should be considered to define relationships between EJBs and maintained appropriately.

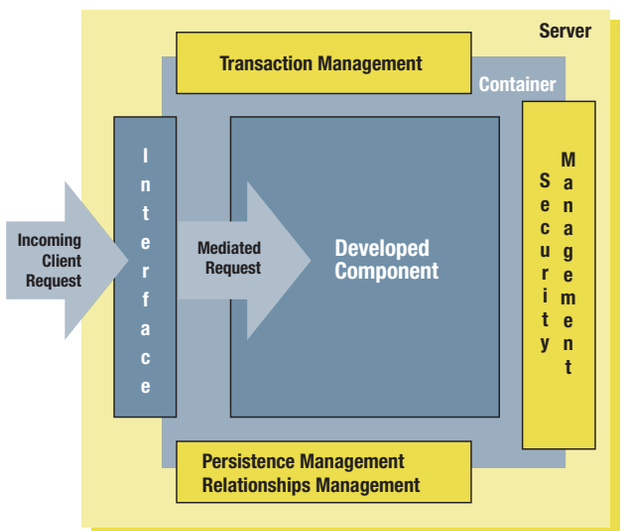

Figure 3 **Control by interposition**

The container is responsible for persistence and relationship maintenance and to provide the ability to search across persistent stores and to give the ability to navigate across relationships; the J2EE defines a new query language, called EJB-QL. This closely follows the spirit of SQL but is defined purely in EJB-property terms, making it at least notionally independent of the underlying persistence mechanism. Figure 4 shows how this is used within an Entity EJB deployment descriptor to add a new tool-created method called findLineItemsGivenQuantity to the EJB.

Note that, being based on XML, deployment descriptors are (once again) ideal for tool-based manipulation.

Sun does not talk solely about technology when defining the J2EE. Its 'Blueprints' site (**http://java.sun.com/blueprints**) provides an extensive list of resources covering J2EE guidelines for implementation, best practices, direct Q&A, discussions of various patterns appropriate to the effective design of J2EE-based systems as well as providing a valuable code resource showing how to develop end-to-end applications.

To aid in identifying the tasks that must be

```
<query>
  <query-method>
    <method-name>findLineItemsGivenQuantity</method-name>
    <method-params>
      <method-param>int</method-param>
    </method-params>
  </query-method>
  <ejb-ql>
SELECT OBJECT (o) FROM Orders AS o
  IN (o.lineItems) li WHERE li.quantity = ?1
  </ejb-ql>
</query>
```

Figure 4 **EJB-QL**

performed by various parties throughout a project's lifecycle, the J2EE platform defines various distinct roles (these are fairly self-explanatory; a more in-depth exposition of these roles is given on Sun's website at: **http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/platform_technologies/platform_roles**) including: J2EE Product Provider, Application Component Provider, Application Assembler, Deployer, System Administrator and Tool Provider. Although a given individual typically performs several roles as a project progresses, the above breakdown attempts to guide the definition of responsibilities and workflow to ensure that all aspects of a system can be considered, regulated and supported by the provision of an appropriate tool set.

Issues

Many people cite performance as their main reason for being wary of the technology. As the J2EE specification matures and as the vendors' offerings become more sophisticated and as people grow to better understand how to apply it, this is one issue that may go away over time. When I went to university, the common mantra was that SQL databases simply would not perform sufficiently well for 'real world' applications. Since that time, SQL has become universal. Performance issues (real or imaginary) have now, for the most part, been devalued. So it will be with J2EE, I believe.

Of greater concern is the potential for fragmentation of the marketplace. While the J2EE is standardised and closely shepherded by Sun, the various vendors and J2EE licensees are introducing sophisticated frameworks that build on top of J2EE: numerous personalisation, commerce and portal frameworks are being introduced at a rate of knots and without the guiding influence (good or bad) of a single standards-setting body. This has the potential to create the same sort of lock-in that has caused suffering since commercial computing first arrived: caveat emptor!

The Future

J2EE is now becoming well-established and is allowing vendors to re-examine their existing products. Macromedia have announced that their popular ColdFusion product will become a value-added service that runs on existing J2EE application servers such as those from IBM, Sun, BEA, Oracle, and Macromedia's own JRun. IBM has also recently announced that it will be undertaking a similar migration for Lotus Notes, which will closely integrate with its Websphere J2EE application server.

The only potential external threat to J2EE's continued wellbeing is from Microsoft's new .NET technology. At this stage, it is impossible to gauge how much impact this will have.

One key positive indicator for J2EE is the existence of an extremely active and vital community of users. This is reflected in both the growing number of application servers and commercial products appearing in the marketplace and in the growing amount of community-sponsored development (products, tools and design techniques) that is taking place.

J2EE is currently enjoying strong commercial and community support. The number of J2EE projects being undertaken both internationally and within Australia is growing. Now is the time to investigate whether J2EE is an appropriate tool for tackling your organisation's needs. ■

FURTHER INFORMATION

http://www.theserverside.com, a very useful site for understanding latest developments in the J2EE and server-side infrastructure in general.

Nicholas Kassem, Enterprise Team, Designing Enterprise Applications with the Java™ 2 Platform, Enterprise Edition, Addison Wesley, 2000 (ISBN 0-201-70277-0).

http://java.sun.com/j2ee, the definitive J2EE site.