microCAD 2005, 19th International Scientific Conference, University of Miskolc Hungary.
pp. 327-334.

Application Servers in E-Commerce Applications

Péter Mileff[1], Károly Nehéz[2]
[1]PhD *student,* [2]PhD, *Department of Information Engineering, University of Miskolc*

Abstract

Nowadays there is a growing demand to develop distributed enterprise level software applications and leverage the flexibility, speed, security and reliability of latest server-side technologies. The Java2 Platform Enterprise Edition (J2EE) technology provides a component-based approach to design, develop, assembly and deploy of enterprise applications. The J2EE platform offers a multi-tiered distributed application model, the ability to reuse components, integrated XML-based data exchange, a security model, and flexible transaction control for application developers.

This paper presents an introduction to basic concepts and terms and main advantages of J2EE technology based application servers.

Keywords: **Java technology**, **JMX, J2EE, JBoss**

1. Introduction to J2EE technology

The J2EE platform introduces a multi-tier, distributed application model. Application logic can be divided into several tiers according to functions. Typically each modern enterprise level software application can be separated logically into minimal three tiers: client, business logic, and database tier. Theoretically each tier can be represented on the same or different servers that lead a very flexible architecture.

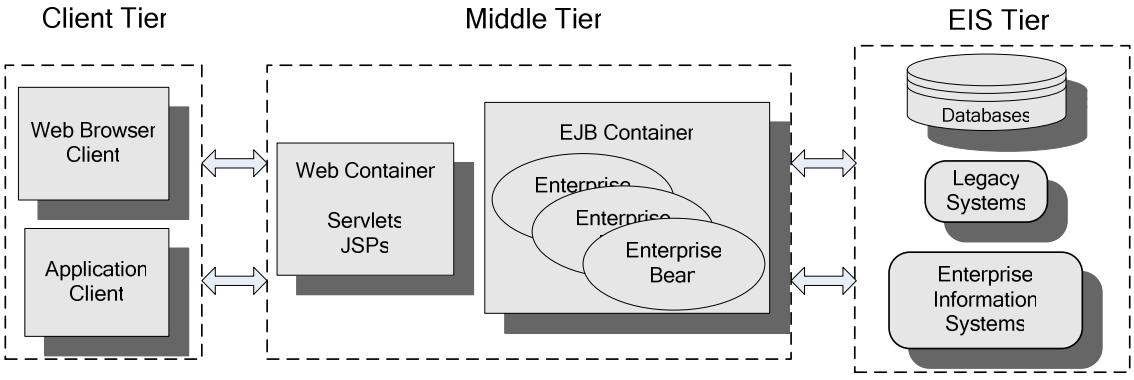

Figure 1.
J2EE Support for three tier applications

The *client tier* is responsible for data representation. It includes browser-based or standalone clients and even legacy client applications. The *middle tier* supports components, modules that provide application services to the client and that implement the application's business logic [3]. The *enterprise information system* tier handles enterprise information

system software and includes enterprise infrastructure systems such as enterprise resource planning (ERP), mainframe transaction processing, traditional relational or modern object oriented database systems, and other legacy information systems. Functional decomposition of these three layers makes it possible to naturally separate logical layers (*data-model*, *business logic* and *representation layers)* of a software application.

1.1 J2EE services

The J2EE framework allows developing distributed applications by providing a set of basic services around transaction management, security checks, state management, and resource management. It is the application server that provides the actual implementation of the J2EE Framework. The following table describes the technologies used by J2EE to implement parts of a distributed application:

Technology	Description
Java Database Connectivity (JDBC)	A standard API used to connect the Java platform to database resources in a vendor-independent manner.
RMI-JRMP	Standard Java Remote Method Invocation (RMI) that uses the Java Remote Message Protocol (JRMP) to implement remote process communication in a networked infrastructure.
Java Interface Definition Language (Java IDL)	A service that incorporates Common Object Request Broker Architecture (CORBA) into the Java platform to provide interoperability using standard Interface Definition Language (IDL) defined by the Object Management Group (OMG). [8]
Remote Method Invocation-Internet Inter-ORB Protocol (RMI-IIOP)	Protocol that enables RMI programmers to combine the benefits of using the RMI APIs and CORBA IIOP communications protocol to communicate with CORBA-compliant clients that have been developed using any language compliant with CORBA. [8]
Enterprise JavaBeans (EJB)	Component architecture for the development and deployment of component-based distributed business applications.
Servlet technology	Open technology to extend functionalities of java based web servers. Based on a request-response mechanism with a Web clients.
JavaServer Pages (JSP)	XML based scripting technology, for building applications containing dynamic Web content, using Java as a scripting language.
Java Message Server (JMS)	An API to communicate with Message Oriented Middleware (MOM) to enable point-to-point and publish/subscribe messaging between systems.
Java Naming and Directory Interface (JNDI)	A unified interface to access different types of naming and directory services.
Java Transaction API (JTA)	A set of APIs that allows transaction management. Applications can use the JTA APIs to commit and roll back (abort) transactions.
JavaMail	An API that provides a platform-independent and protocol-independent framework to build mail and

	messaging applications.
JavaBeans Activation Framework (JAF)	API for an activation framework that is used by other packages, such as JavaMail. JAF is used to determine the type of data, encapsulate access to that data, discover the operations available on that data, and instantiate the appropriate bean to perform these operations. For instance: JavaMail uses JAF to determine what object to instantiate based on the type of the attached object of an email.

1.2 Components and Containers

Conceptually the J2EE architecture divides the programming environment into containers. A *container* is a standardized runtime environment that provides specific services to components. *Container* is an interface between components and the low-level platform specific functionality that supports the component. A *component* is an application-level software unit supported by a container that can be reused by other enterprise applications. Technically speaking, a component is a reusable piece of software that encapsulates *data and behaviour*, has a defined life-cycle model, and provides services to clients. In the J2EE environment, a Web container provides standard Web-specific services, whereas an EJB container provides services to support enterprise bean components. For example, an EJB container provides such services as transaction management, security, multi-threading, distributed programming, and connection pooling. A Web container provides communication APIs and protocols and network services to facilitate sending and receiving requests and responses. Figure 2 illustrates the J2EE component types and their containers.

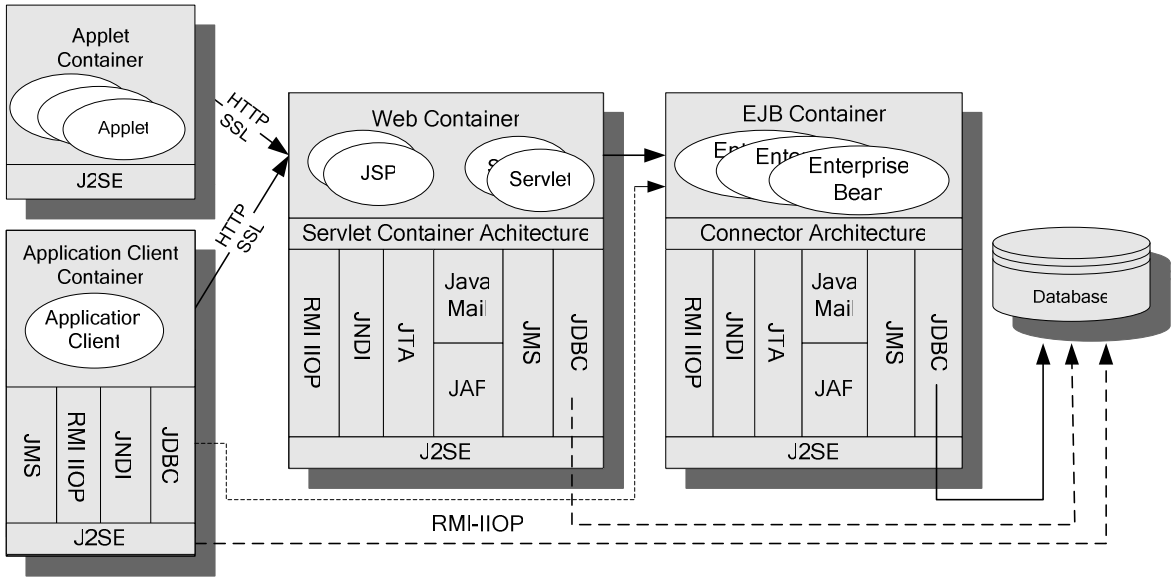

Figure 2.
J2EE Components and Containers

The J2EE client tier supports a variety of client types. A client may be a Web browser using HTML pages, or it may use dynamic HTML generated with JSP technology. A client

may be a Java applet or a standalone Java application. J2EE clients access the middle tier using standard Web communication protocols. In multi-tier environments, they never directly access the EIS tier.

The middle tier consists of the Web and EJB containers, plus other services, such as Java Naming and Directory Interface (JNDI), JMS, JavaMail, and etc. The Web container provides the programming environment for developing and deploying servlets and JSPs. Typically, servlets and JSPs encompass an application's presentation logic and the logic that controls client interaction. Web components, when packaged together, comprise a Web application. The Web container, through servlets and JSPs, provides runtime support for receiving HTTP requests and composing HTTP responses to these requests. It ensures that results are returned to the requesting client [2]. The EJB container, which is also in the middle tier, provides the environment for developing and running enterprise bean components. It is often considered the backbone of the J2EE programming environment. Enterprise bean components are Java code that implement an enterprise's business processes and entities. They perform the application's business operations and encapsulate the business logic. The EJB container automatically handles transaction and life-cycle management for its enterprise bean components. In addition, the EJB container provides other services to its beans, such as lookup and security services, and standardized access via the Connector architecture to the EIS tier database or legacy system.

2. Application servers

Application servers are *middleware* platforms for development and deployment of component-based software. Application server offer an environment in which users can deploy application components - software components, developed either by the users themselves or by third-party providers that correspond to server-side parts of distributed applications. Most application servers implement one of the industry standards currently adopted for server-side application components: J2EE, .NET or the CORBA Component Model.

2.1 JBoss Application server

JBoss is an extensible, dynamically configurable Java based application server which includes a set of J2EE compliant components. JBoss is an open source middleware, in the sense that users can extend middleware services by dynamically deploying new components into a running server. The Java Management Extensions (JMX) specification [4], provides the base of JBoss middleware components. JMX defines architecture for dynamic management of resources distributed across a network. In JMX, as in other management architectures offers a dynamic management of components. Dynamic management means that container is able to dynamically load, unload components, without stopping the applications. JMX was chosen as the basis of the JBoss component model for the following advantages: it provides a lightweight environment in which components can be dynamically loaded and updated; it supports component introspection and component adaptation; it decouples components from their clients; it can be used as a realization of the microkernel architectural pattern. The JBoss service component model extends and refines the JMX model to address some issues beyond the scope of JMX: service lifecycle, dependencies between services, deployment and

redeployment of services, dynamic configuration and reconfiguration of services, and component packaging. Service components implement every key feature of J2EE: naming service, transaction management, security service, servlet/JSP support, EJB support, asynchronous messaging, database connection pooling, and IIOP support. They also implement important features not specified by J2EE, like clustering and fail-over.

JBoss supports a generalization of the EJB model by using service components as meta components. Its meta-level architecture for generalized EJBs is built upon four kinds of elements: *invokers*, *containers*, *dynamic proxies*, and *interceptors*. *Invokers* are service components that provide a general remote method invocation service over a variety of protocols. *Containers* are service components that enhance application component classes with predefined and packaged sets of aspect requirements. They provide serverside join points for aspects that crosscut the central concerns of multiple EJB components. *Dynamic proxies*, used as client stubs, provide similar join points at the client side. *Interceptors* implement crosscutting aspects at both sides. Containers, proxies, and interceptors are neither created nor manipulated by initiatives of the server spine, but by actions of an EJB deployer, which is a service component itself. In other words, EJB support is pluggable [6].

2.2 JBoss JMX Architecture

The JMX architecture is shown in Figure 3. It consists of three levels: the *instrumentation*, the *agent*, and the *distributed services* level. The *instrumentation* level defines how to instrument resources so that they can be monitored and manipulated by management applications. The instrumentation of a given resource is provided by one or more managed beans (MBeans), Java objects that conform to certain conventions and expose a management interface to their clients.

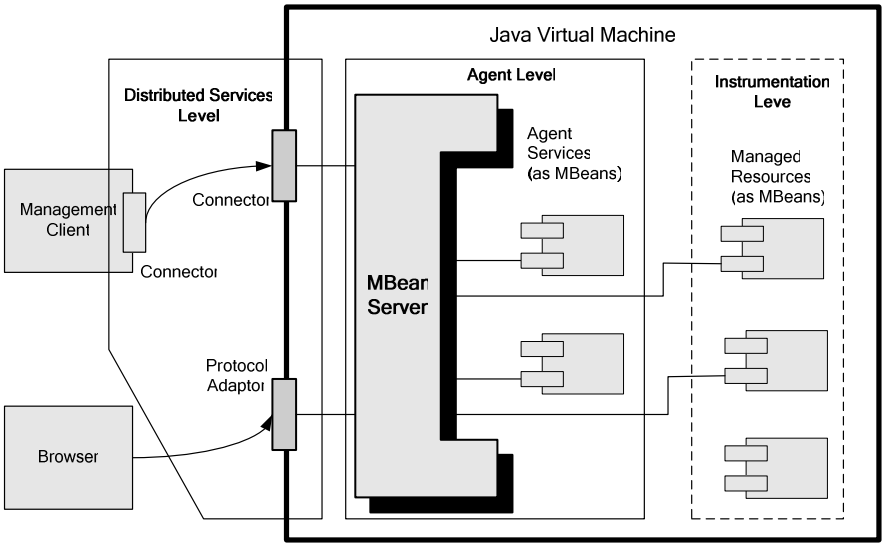

Figure 3.
The JBoss-JMX architecture

The *agent level* defines an agent that manages the set of instrumented resources within a Java virtual machine, in behalf of (possibly remote) management applications [6]. The JMX agent

consists of an in-process server, the MBean server, plus a standardized set of agent services: dynamic class loading, monitoring, timer service, and relation service. Agent services are implemented as MBeans; this makes them manageable through the MBean server, like user resources. The *distributed services level* specifies how management applications interact with remote JMX agents and how agent-to-agent communication takes place. It consists of connectors and protocol adaptors, implemented as MBeans. This level is not fully defined at the present phase of the JMX specification process. Together, the instrumentation and agent levels define an in-process component model. The MBean server provides a registry for JMX components (MBeans) and mediates any accesses to their management interfaces [1,6,7].

3. Conclusion

A new approach to design and implement enterprise level software applications has been presented. Continuing work will focus on setting up a small J2EE environment and simulate its scalability and performance with a special self developed J2EE application.

4. Acknowledgements

The authors are grateful to the colleagues of the Department of Information Engineering at University of Miskolc. Special thanks are due to Production Information Engineering and Research Team (PIERT) established at the Department of Information Engineering and supported by the Hungarian Academy of Sciences for the financial support of the research.

5. References

[1] J. Lindfors, M. Fleury, The JBoss Group: JMX: *Managing J2EE with Java Management Extensions*. SAMS Publishing Inc., 2002.

[2] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann: *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. Wiley, 2000.

[3] Sun Microsystems: Enterprise JavaBeans Specification, Version 2.0, java.sun.com/products/javabeans, 2003.

[4] Sun Microsystems: *Java 2 Platform Enterprise Edition Spec.*, java.sun.com/products/javabeans, 2003.

[5] E. Checchet, J. Marguerite, W. Zwanepoel: *Performance and scalability of EJB applications. In Conference on Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA'02), 2002.

[6] Marc Fleury, Francisco Reverbel: The JBoss Extensible Server, http://citeseer.ist.psu.edu/697000.html , 2003.

[7] Rod Johnson Wrox: *Expert One-on-One J2EE Design and Development,* Willey Publishing Inc. 2003.

[8] Object Management Group: *Corba spectification*, www.omg.org, 2004

microCAD 2005, 19th International Scientific Conference, University of Miskolc Hungary.
pp. 327-334.