Java 2 Platform, Enterprise Edition (J2EE): Enabling Technologies for EAI

Tony Ng, Staff Engineer
Rahul Sharma, Senior Staff Engineer
Sun Microsystems Inc.
J2EE Overview

Tony Ng, Staff Engineer
Sun Microsystems Inc.
What is Java 2 Platform, Enterprise Edition?

• Java 2 Enterprise Edition (J2EE) is a platform that enables solutions for developing, deploying and managing n-tier server-centric enterprise applications.

• J2EE provides building blocks for EAI
Java™ Platforms

Java Technology Enabled Devices

Java Technology Enabled Desktop

Workgroup Server

High-End Server

Micro Edition

Standard Edition

Enterprise Edition
J2EE Platform Deliverables

• J2EE specification
  – Defines the J2EE requirements
• J2EE reference implementation
  – Prototype of J2EE platform
• J2EE compatibility test suite
  – Validates J2EE platform compatibility
• J2EE blueprints
  – Best practices for J2EE solutions
J2EE Reference Implementation

- Fully functional prototype
- Allow development of compatibility tests
- Help developers prototype J2EE technology-enabled applications
- Help server platform and tool vendors build J2EE platform support in their products
- Source code available
Compatibility Test Suite

- Allow branding of compatible servers
- Ensure all J2EE implementations branded by Sun conform to the J2EE specification
- Move us towards the goal of Write Once, Run Anywhere
J2EE Blueprints

• Best practices for J2EE
  – Examples, patterns, templates
• J2EE blueprints include:
  – Book, *Designing Enterprise Applications*
  – Application— “Java Pet Store”
J2EE Components

- Enterprise JavaBeans™ (EJB™)
  - Business logic
- JavaServer Pages™ (JSP)
  - Dynamic content
- Servlet
  - Portable CGI script
- Application Client
  - First-tier Java application
J2EE Technologies

• J2EE Connector Architecture
  – Enterprise systems integration

• Java Message Service (JMS)
  – Asynchronous messaging

• Java API for XML Processing
  – XML transformation

• Java Transaction API
  – Data integrity
J2EE Technologies (Cont...)

- Java Naming and Directory Interface (JNDI)
  - Enterprise directory
- JavaMail
  - Email Notification
- JDBC
  - Database connectivity
- JavaIDL, RMI, CORBA
  - Remote services
J2EE Containers

• Container handles:
  – Concurrency
  – Transactions
  – Security
  – Scalability
  – Distribution

• Components provide:
  – Business logic
  – Presentation
J2EE Components and Containers

- Applet Container
  - Applet
  - J2SE

- App Client Container
  - App
  - HTTP/HTTPS

- Web Container
  - JSP
  - Servlet
  - J2SE
  - JNDI
  - JMS
  - JTA
  - JavaMail
  - JAF
  - RMI/IIOP
  - JDBC

- EJB Container
  - EJB
  - JNDI
  - JMS
  - JTA
  - JavaMail
  - JAF
  - RMI/IIOP
  - J2SE

- Database

- HTTP/HTTPS
Benefits of Using J2EE

• Faster solution delivery time to market
  – Components
  – JavaServer Pages, EJB and other APIs

• Freedom of choice
  – No vendor lock
  – Skill reuse

• Simplified connectivity
  – JDBC, Connector, CORBA, JMS
J2EE and EAI

- Components and containers:
  - EJB: business process
  - Servlet, JSP: web components
- JAXP: XML parsing and transformation
- Connectors: EIS integration
- JMS: Asynchronous messaging
- Future: JAXM, XML data binding
Building EAI framework

• J2EE provides building blocks
• EAI frameworks add:
  – Intelligent routing
  – Metadata facility
  – Rules engine
  – Integration of business processes
  – Workflow
  – Tools
Enterprise JavaBeans

Tony Ng, Staff Engineer
Sun Microsystems Inc.
The Problem

- Difficult to write distributed, n-tier applications
- Focus on business logic, not plumbing
- Multi-platform, heterogeneous environment
- Time to market
What Is the Enterprise JavaBeans™ Architecture?

• A component architecture for the development and deployment of distributed business applications

• Goals
  – Simplify development and deployment of business applications
  – Separate business logic from system code
  – Portable components (Write Once, Run Anywhere™)
  – Allow multi-vendor interoperability
EJB Features

- Flexible component model
- Attribute-based application semantics
- Transaction: Declarative and programmatic
- Security: ACLs on operations
- Distributed access: home and remote interfaces
- Deployment: ejb-jar file
Benefits

- Simplifies development
- Write once, run anywhere
- Industry support
- Vendor/platform independent
- Partitioning of expertise
Types of Enterprise Beans

• Session Beans
  – Stateless
  – Stateful

• Entity Beans
  – Bean-managed persistence
  – Container-managed persistence

• Message-Driven Beans
Session Beans

- On behalf of a single client
- May keep conversation state (stateful)
- Relatively short-lived: client session
- Do not survive server crashes
- Example: shopping cart
Entity Beans

- Represents business entities
- Shared access for multiple users
- Long-lived: as long as data in database
- Transactional
- Survive server crashes
- Container Managed Persistence:
  - Relationships, EJB-QL
- Example: Account, Employee
Message-driven Beans

• New enterprise bean type
  – Asynchronous
  – Activated upon message arrival
  – Stateless
  – No home or remote interface
  – Configured as listener for queue/topic
Message-driven Bean Scenario
Roles

- Enterprise bean provider
- Application assemblers
- Deployers
- System administrator
- EJB server/container provider
Deployment Descriptor

- Declarative rather than programmatic
- Allows easy customization
- Transaction attributes
- Security attributes
- Access control list
- Resource and EJB references
- Environment properties
Transaction

- Declarative or programmatic
- Transparent support for distributed transaction
- EJB server handles
  - transaction boundaries
  - propagation
  - resource enlistments
Security

- Declarative and/or programmatic
- Access control list on business methods
- RunAs mode
- Security-related methods in enterprise bean context
Writing Enterprise beans

- Implement business methods and component contract
- Define home and remote interface
- Define deployment descriptor
- Deploy and Run
- Tools support
Home Interface

- Allows a client to create, locate or destroy a bean

- Example:

```java
public interface AccountHome extends EJBHome {
    public Account create(String ssn, String name)
        throws CreateException, RemoteException;

    public Account findByPrimaryKey(String ssn)
        throws FinderException, RemoteException;
}
```
Remote Interface

- Exposes business methods to clients
- Example:

  ```java
  public interface Account extends EJBObject {
    public void debit(double amount) throws InsufficientBalanceException, RemoteException;
    public void credit(double amount) throws RemoteException;
    public double getBalance() throws RemoteException;
  }
  ```
public class AccountBean implements EntityBean {
    // entity state
    private String name;
    private double balance;
    public void debit(double amount) throws InsufficientBalanceException {
        if (balance - amount < 0.0) {
            throw new InsufficientBalanceException();
        }
        balance -= amount;
    }
    // more business methods and
    // component contract implementation
    ...
}
XML Support in J2EE

Tony Ng, Staff Engineer
Sun Microsystems Inc.
XML in J2EE

• Java API for XML Processing
• Future Technologies:
  – Java API for XML Messaging (JAXM)
  – Java XML Data Binding
JAXP 1.1

• Java Community Process initiative JSR-000063
• Feature list:
  – SAX2.0.
  – SAX 2 extensions.
  – DOM Level 2 Core interfaces.
  – XSLT 1.0
  – Improved mechanism for factory lookup.
JAXP: Processing APIs for Java

User Application

JAXP Interfaces

Reference Impl  Other Impl
• A lightweight API for parsing XML Documents
• API for transformation, referred to as TrAX.
• Allows for pluggable parsers
• Allows for pluggable XSLT engines.
• Allows processing of XML using:
  – Callback Driven (SAX)
  – Tree Based (DOM)
  – XSL Transformation.
SAX2.0 API

- **Simple API for XML**
- Accesses Documents Serially
- Fast and Lightweight
- Harder to Program
- Sequential Access Only
- `Org.xml.sax`
- `Org.xml.sax.ext`
SAX 2.0 (Contd)

XML Document → Parser → Events → Provided Handler
import java.xml.parsers.*
import org.xml.sax.*

SAXParserFactory factory =
    SAXParserFactory.newInstance();
factory.setValidating(true);
SAXParser parser = factory.newSAXParser();
parser.parse("http://server.com/foo.xml", handler);

// can also parse InputStreams, Files, and
// SAX input sources.
import org.xml.sax.*

public class MyHandler implements ContentHandler {
    public void startDocument() throws SAXException {
        // start processing here. Could store it in a
        // stack and build your own tree representation
        // This is the first method invoked
    }

    public void endDocument() throws SAXException {
        // This is the last method invoked on the
        // handler
    }
}
DOM Level 2

- Document Object Model
- Access XML document via a tree structure
- Composed of element nodes and text nodes
- Can "walk" the tree back and forth
- Larger memory requirements
- `org.w3c.dom.*`
DOM Level 2 (Contd)
import java.xml.parsers.*
import org.w3c.dom.*

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setValidating(true);
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc =
        builder.parse("http://server.com/foo.xml");

// can also parse InputStreams, Files, and // SAX input sources.
XSLT 1.0

• XSL Transformations.
• Syntax and semantics for transforming XML documents into other XML documents or any other format.
• Expressed as a well-formed XML document.
• Recommended Processing Instruction to apply to an XML document
  – `<?xml-stylesheet href="mystyle.css" type="text/css"?>`
XSLT 1.0 (Contd)

XML Document → XSLT Processor → Resulting XML Document

Input

XSL Stylesheet

Result
import java.xml.transform.*;

Transformer transformer;
TransformerFactory factory =
TransformerFactory.newInstance();
try {
    // Create a transformer for a particular stylesheet.
    transformer = factory.newTransformer(
        new StreamSource(stylesheet));

    // Transform the source xml to System.out.
    transformer.transform(new StreamSource(sourceId),
        new StreamSource(System.out));
} catch (Exception e) {
    // handle error
}
The following example illustrates the serialization of a DOM node to an XML stream.

```java
import java.xml.transform.*;

TransformerFactory tfactory = TransformerFactory.newInstance();
Transformer serializer = tfactory.newTransformer();
Properties oprops = new Properties();
oprops.put("method", "html");
oprops.put("indent-amount", "2");
serializer.setOutputProperties(oprops);
serializer.transform(new DOMSource(doc),
new StreamResult(System.out));
```
Plugability

- Factory lookup is accomplished by
  - System property
    - `javax.xml.parsers.SAXParserFactory`
    - `javax.xml.parsers.DocumentBuilderFactory`
    - `javax.xml.transform.TransformerFactory`
  - `$JAVA_HOME/lib/jaxp.properties` file
  - Jar Services API
    - `META-INF/services/javax.xml.parsers.XXXFactory`
  - Reference Default
Reference Implementation

- Parser and XSLT Engine based on ASF Code
- Apache Crimson for XML Parsing
- Apache Xalan for XSLT Processing
Current Status

• Public Review 2 of the spec and beta of the RI can be obtained by following links from
  – http://java.sun.com/xml

• feedback aliases
  – jsr63-comments@eng.sun.com