What's New in the Java™ Platform, Enterprise Edition 5 and Beyond

Roberto Chinnici
Senior Staff Engineer
Sun Microsystems, Inc.

roberto.chinnici@sun.com
Contents

- Java™ Platform, Enterprise Edition 5
- Guiding theme
- New technologies
- Updates to existing ones
- How can you get a preview?
- The future
Caveat

- All new Java™ API go through the Java Community Process
- Java™ Platform, Enterprise Edition 5 ("Java™ EE 5") is not final yet
- Anything you see here may still change
- On the plus side, if you send us your feedback, it may change the way YOU want!
Main Theme for Java™ EE 5

Ease of Development
Why Ease of Development?

- Java™ EE is very powerful
- Industry standard for robust enterprise applications
- But many developers find it difficult to get started
- Need lots of boring boilerplate
- We can make it simpler!
Developer Segmentation

- Enterprise
  - Composite Modeling & Integration
  - Transactional Business Logic (3 tier)
  - Applications: data access + web services (2 tier)
  - Applications with data access
  - Web pages: + dynamic content
- Corporate
  - Web pages

Content

App Complexity
Goals

- Make Java™ EE programming easier to learn
- Improve developer productivity
- Enable better tools
- Attract new developers to Java™ EE
- Expand the Java™ EE market
Major Areas We Focused On

- Business logic
- Web services
- Persistence
- Web apps
Major Areas We Focused On

- Business logic – EJB 3.0
- Web services – JAX-WS 2.0, JAXB 2.0, JSR-181
- Persistence – EJB 3.0
- Web apps – JSF 1.2, JSTL 1.2
The Challenge

- It's hard to make something simpler by **adding** things to it
- Yet compatibility with J2EE™ 1.4 is essential
- All existing applications must still run
- Moreover, migration to the new technologies should be as painless as possible
Some of the Complaints

- Very powerful, but complex to use
- Too many classes, interfaces
- JNDI lookups everywhere
- Checked exceptions galore
- Deployment descriptors
- Entity beans
EJB 2.1 Example

```java
public class PayrollBean
    implements javax.ejb.SessionBean {

    SessionContext ctx;
    DataSource empDB;

    public void setSessionContext(SessionContext ctx) {
        this.ctx = ctx;
    }

    public void ejbCreate() {
        Context initialContext = new InitialContext();
        empDB = (DataSource)initialContext.lookup("java:comp/env/jdbc/empDB");
    }
```
public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbRemove() {}

public void setBenefitsDeduction(int empId,
    double deduction) {
    ... 
    Connection conn = empDB.getConnection();
    ...
    }
    ...
}
EJB 2.1 Example (Continued)

```xml
<session>
  <ejb-name>PayrollBean</ejb-name>
  <local-home>PayrollHome</local-home>
  <local>Payroll</local>
  <ejb-class>com.example.PayrollBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Container</transaction-type>
  <resource-ref>
    <res-ref-name>jdbc/empDB</res-ref-name>
    <res-ref-type>javax.sql.DataSource</res-ref-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</session>
...
<assembly-descriptor>...</assembly-descriptor>
```
The Approach We Took

- POJO-based programming
  - More freedom, fewer requirements
- Extensive use of annotations
  - Reduced need for deployment descriptors
- Lots of defaults
- Dependency Injection for resources
  - Inversion of control
- New APIs and frameworks
EJB 3.0 Example

@Stateless
public class PayrollBean {

    @Resource SessionContext ctx;
    @Resource DataSource empDB;

    public void setBenefitsDeduction(int empId, double deduction) {
        ...Connection conn = empDB.getConnection();...
    }

    ...}

}
Annotations

• Making extensive use of annotations
  • For defining components
  • To specify external dependencies
  • For defining and using web services
  • To map Java™ classes to XML
  • To map Java™ classes to databases
  • To reduce need for deployment descriptors

• Just starting to scratch the surface of what is possible
Not As Easy As It Seems

- Annotations seem intuitive:
  ```java
  @Resource DataSource empDB;
  ```
- But there is magic involved:
  - Who's setting the field?
  - When?
  - But the field is package-private!
  - And where does the data source come from?
- Need uniform rules, clear processing and runtime model, good error reporting
Defaults Everywhere

- Descriptors forced developers to write trivial stuff over and over again
- Annotations have meaningful defaults
- In many cases, deployment tools will figure out things on their own
  - e.g. References to components or web services in the same application
- Visible information is relevant
Tour of Java™ EE 5

- Enterprise JavaBeans™ 3.0
  - Components and persistence
- Web Services
  - JAX-WS 2.0, JAXB 2.0, JSR-181, JSR-109
- Web tier
  - JSF 2.1, JSTL 1.2
- Other updates
  - StAX, JSP, Servlets, JavaMail, ...
EJB 3.0 – Simplified API

- No required container interfaces
- No required deployment descriptor
- Plain objects, plain interfaces
- No checked exceptions
- Dependency injection and JNDI-less lookups
- Interceptors
From a Developer's POV

- New-style EJB components are plain objects
- They implement plain interfaces
- They declare dependencies using annotations
- Transactions and security are declarative too
- Container provides services via interposition
- Developer can join in using lifecycle events and interceptors
Example – Bean

@Stateless
public PayrollBean implements Payroll {

    @TransactionAttribute(MANDATORY)
    public void setBenefitsDeduction(int empId, double deduction) {...}

    public double getBenefitsDeduction(int empId) {...}

    public double getSalary(int empId) {...}

    @TransactionAttribute(MANDATORY)
    @RolesAllowed("HR_PayrollAdministrator")
    public void setSalary(int empId, double salary) {...}

}
Example – Client

// inside another component

@EJB
private Payroll payroll;

public double computeBonus(int empid) {
    return payroll.getSalary(empid) * BONUS_FACTOR;
}

...
Interoperability and Migration

- Applications written to EJB 2.1 and earlier work unchanged in EJB 3.0 containers
- Migration path to EJB 3.0 APIs
  - New applications can be clients of older beans
  - Older clients can be clients of new EJB 3.0 components
- Many EJB 3.0 ease-of-use features available to components written to EJB 2.1 view
  - Injection, interceptors, defaults, annotations, ...
EJB 3.0 – Persistence

- Resolved EJB/JDO conflict!
  - Single persistence API
  - Defined in a separate specification developed by the EJB 3.0 expert group
  - Persistence API also usable in non-Java™ EE environments

- Builds on years of experience with existing technologies and products
- Much simpler than EJB 2.1 CMP
Persistence Model

- Entities are plain Java™ classes
  - Concrete classes – support use of “new”
  - JavaBeans™ properties or persistent fields
  - No required bean or callback interface
  - Inheritance supported
- Usable as detached objects in other application tiers – no more DTOs
- Usable across multiple transactions, spanning multiple user requests
Example – Entity

```java
@javax.persistence.Entity(access=FIELD)
public class Person {
    @javax.persistence.Id
    protected String name;

    protected int count;

    public void incrementCount() {
        count++;
    }
}
```
Example – Entity Client

@Stateless
class Hello {
    @Resource
    private EntityManager em;

    public String sayHello(String name) {
        Person p = em.find(Person.class, name);
        p.incrementCount();
        return "Hello " + param;
    }
}
Web Services

- New stack based on JAX-WS 2.0 and JAXB 2.0
- Annotation-based, easy to use
- Support for latest standards:
  - SOAP 1.2, MTOM/XOP, all of XML Schema 1.0
  - WS-I Basic Profile 1.1, Attachment Profile 1.0
- More WS-* technologies in the pipeline:
  - WS-Addressing (JSR-261)
  - WS-Policy (JSR-265)
JAX-WS 2.0

- Successor to JAX-RPC 1.1
- Annotation-based, boilerplate-free
- Protocol- and transport-independent
- Message-based interactions
- Asynchronous web services
- Standard customization language
- Supports SOAP 1.1/1.2 and RESTful services
Example – Web Service (1)

```java
@WebService
@Stateless
public class CreditRating {
    public Score getCredit(Customer c) {...}
}
```
Example – Web Service (2)

```java
@WebService(name="CreditRatingService",
targetNamespace="http://example.org")
@Stateless
public class CreditRating {

    @WebMethod(operationName="getCreditScore")
    public Score getCredit(
        @WebParam(name="customer")
            Customer c) {...}

}
```
Example – Client

```java
@Stateless
class Hello {

    @WebServiceRef(CreditRatingService.class)
    private CreditRating rating;

    public Rating rate(Customer c) {
        Score s = rating.getCredit(c);
        return asNormalizedRating(s);
    }
}
```
Example – Asynchrony

// generated client interface
@WebService
public interface CreditRatingService {

    // sync operation
    Score getCreditScore(Customer customer);

    // async operation w/polling
    Response<Score>
    getCreditScoreAsync(Customer customer);

    // async operation w/callback
    Future<?>
    getQuoteAsync(Customer customer,
    AsyncHandler<Score> handler);

}
Dispatch and Provider API

- Low-level messaging API
- Choose the best message representation:
  - `javax.xml.transform.Source`, JAXB object, `javax.xml.soap.SOAPMessage`, ...
- Build gateways/dispatchers yourself
- Don't reinvent the wheel:
  - Reuse JAX-WS protocol bindings, transports, message handlers, message-level security
Dispatch Interface

// T is the type of the message
public interface Dispatch<T> {

    // synchronous request-response
    T invoke(T msg);

    // async request-response
    Response<T> invokeAsync(T msg);
    Future<?> invokeAsync(T msg, AsyncHandler<T> h);

    // one-way
    void invokeOneWay(T msg);
}

Provider Interface

// T is the type of the message
public interface Provider<T> {

    T invoke(T msg);
}


Packaging a Web Service

- JSR-109 descriptor optional, rely on defaults and annotations instead

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<webservices xmlns='http://java.sun.com/xml/ns/j2ee'version='1.1'>
  <webservice-description>
    <webservice-description-name>HelloService</webservice-description-name>
    <wsdl-file>WEB-INF/wsdl/HelloService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>WEB-INF/HelloService-mapping.xml</jaxrpc-mapping-file>
  </webservice-description>
</webservices>

<?xml version='1.0' encoding='UTF-8' ?>
<configuration xmlns='http://java.sun.com/xml/ns/jax-rpc/ri/config'>
  <service name='HelloService' targetNamespace='urn:HelloService/wsdl' typeNamespace='urn:HelloService/types' packageName='endpoint'>
    <interface name='endpoint.HelloServiceSEI' servantName='endpoint.HelloServiceImpl'>
    </interface>
  </service>
</configuration>
```
JAXB 2.0

- Entirely annotation-based
- Supports 100% of XML Schema 1.0
- Mapping of XML Schema to Java™ classes and vice versa
- Schema evolution
- Partial updateable binding
All of XML Schema!

- Extensibility constructs:
  - Type substitution, element substitution, wildcard
- Built-in datatypes, using new JAXP 1.3 classes
  - e.g. javax.xml.datatype.XMLGregorianCalendar
- Enumerations and all other facets
- Identity constraints and xs:redefine
- Validation via JAXP 1.3
Schema Evolution

- Evolving schemas is hard
  - Naïve solutions run into the UPA constraint
- Lots of debate in the XML community on how to address this (fencepost elements, relaxed wildcards, ...)
- JAXB 2.0 does flexible unmarshalling:
  - Unknown or out-of-order content is ignored
  - If a wildcard is present, it captures the unknown content for processing by the application
Partial Updateable Binding

DOM Subtree could be identified by JAXP 1.3 XPath or StAX

JAXB Binder
associate DOM and JAXB views

XML Document
Load
Save

DOM
Unmarshal
Update

Modify

JAXB Managed Objects
Dynamic Web Service Client

JAXBContext jc = JAXBContext.newInstance("com.acme.foo");
Service svc = Service.create(serviceName);
svc.addPort(portName, SOAP11HTTP_BINDING, address);
Dispatch<Object> d = svc.createDispatch(
    portName,
    jc,
    Service.Mode.PAYLOAD);

AcmeFooRequest request = new AcmeFooRequest();

// ... stuff some data into the request ...

AcmeFooResponse response = (AcmeFooResponse)
    d.invoke(request);

// ... process the response ...
Web Tier

- Very active community
- Already targeting Ease of Development
- Incorporated JavaServer™ Faces 1.2 and JSTL 1.2 into the platform
  - Unified expression language (EL)
- Dependency injection extended to the web tier
- EJB 3.0 persistence usable in the web container
JavaServer™ Faces 1.2

- First class UI component and event model
- Client device independence
- Externalized navigation
- Massively extensible (witness StrutsShale)
- Localizable and accessible
- Tool-friendly
- Reusable component libraries
How Does JSF Fit in the Web Tier?

- API plus tag libraries

![Diagram of Web Tier tiers]

- JSF App
- JSF Tags
- JSP
- JSF API
- Portlet
- Servlets
Ease of Development in JSF 1.2

- Event model familiar to developers, use the JavaBeans™ listener pattern
- Components appear as:
  - JSP custom tags (to a page author)
  - JavaBean objects (to a Java™ developer)
- Binding expressions use the unified EL
- Managed beans instantiated on-demand
Third-Party Component Market

- “VB-effect”
- Renderers and RenderKits
- NavigationHandlers
- Converters and Validators
- AJAX components starting to appear
What Else Is New in Java™ EE 5?

- JSP Standard Tag Library 1.2
- StAX – Streaming API for XML
- Web Services Metadata (JSR-181)
- Common Annotations (JSR-250)
- Lots of updates:
  - JSP 2.1, Servlet 2.5, JavaMail 1.4, JAF 1.1, SAAJ 1.3, ...
Project Glassfish

- Sun Java™ System Application Server 9, Platform Edition
- Open source under OSI-compliant CDDL license
- Java™ EE 5 implementation, final release will be fully compatible
- CVS access, nightly builds, promoted builds
- Configured and built with Maven
Glassfish Community

- Open to external participants
  - Commit privileges for qualified contributors
- Join the community, download the builds, check out the forums and mailing lists, report bugs, submit rfes, contribute code

- http://glassfish.dev.java.net
Annotations put Java™ code editors back as the center point of an IDE

IDE needs to be smart to track dependencies and account for annotation overriding by descriptors

- Visual feedback is critical

New applications will likely stay in the descriptor-free or near-free area

- Beginners won't encounter complex cases
The Future

- Warning: this is pure speculation
Things on the Horizon

- Mustang and Dolphin enhancements
- Portlets
- Java™ Business Integration (JBI)
- AJAX support in JavaServer™ Faces 2.0
- Spring bean factory
- Scripting in the Java™ platform
- Aspect-oriented programming
Summary

- Java™ EE is becoming easier
- No silver bullet
  - Pick our battles
  - Coordinate across multiple JSRs
- Still room for improvement, in Java™ EE 5 we focused on key areas
- Ease of development not a one-time effort