

Java EE 5 Architecture and Update

Lee Chuk Munn

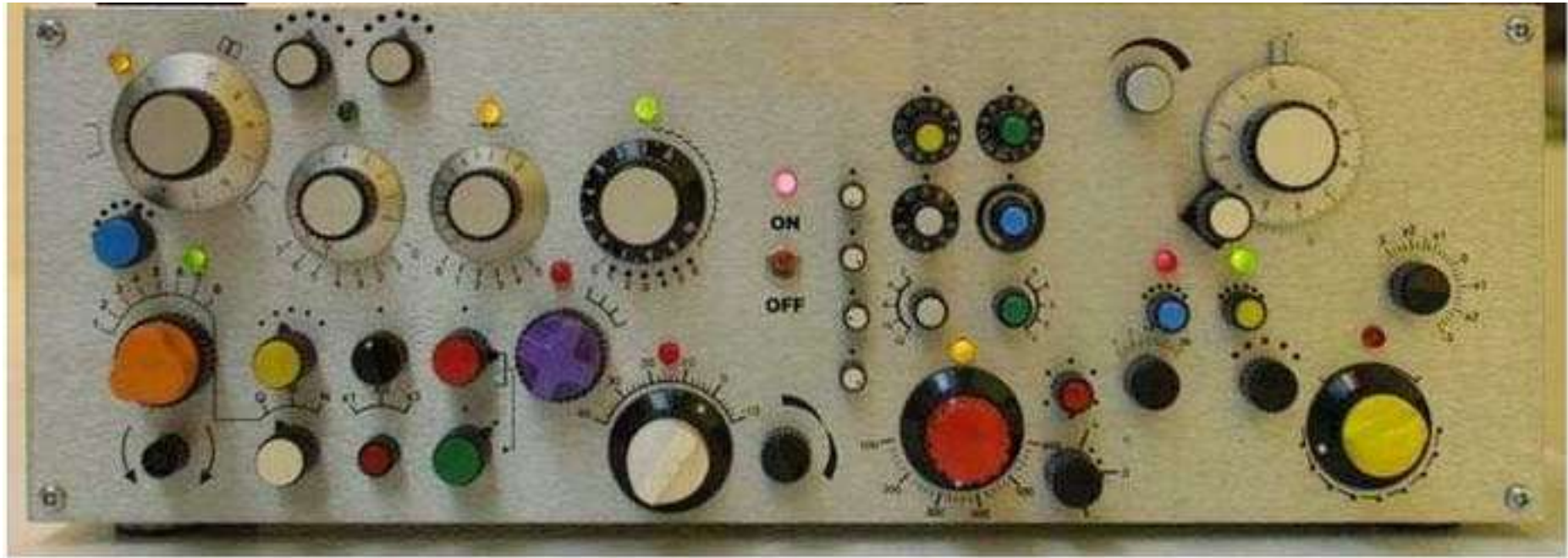
Senior Developer Consultant
Sun Microsystems, Inc.



The J2EE™ Challenge

- J2EE is enormously powerful
 - > The industry standard for robust enterprise apps
- But that power sometimes gets in the way
 - > Too difficult to get started
 - > Even simple apps need boring boilerplate
- Can we keep the power, but make typical development tasks simpler?
- **YES:** and that is the focus of Java EE 5!

The Java EE Platform Today...



Improvements in Java EE 5 Platform

- POJO-based programming
 - > More freedom, fewer requirements
- Extensive use of annotations
 - > Reduced need for deployment descriptors
- Resource Injection
 - > Inversion of control
- New APIs and frameworks

Goal of Java EE 5 Platform...



Java EE 5 Major Features

- Simplified web services support
- More web service standards support
- Greatly simplified EJB™ development
- New persistence API
- Easy web applications with JavaServer™ Faces



EJB 3.0



Primary Goal of EJB 3.0

**Ease of
Development!**

EJB 2.1

- Very powerful, but tedious to use
 - > Too many classes, interfaces
 - > Java Naming and Directory Interface™ (JNDI) API lookups
 - > javax.ejb interfaces
 - > Awkward programming model
 - > Deployment descriptors
 - > Entity bean best practices (anti-patterns)
 - > ...

Ease of Development in EJB 3.0

- Makes EJB technology easier to learn and use
 - > Fewer classes and interfaces
 - > Dependency injection
 - > Simple lookups
 - > No required container interfaces
 - > No required deployment descriptor
 - > Simplified persistence
 - > Object/relational mapping
- Improves developer productivity
 - > Designed to draw new developers to J2EE platform

Ease of Development in Java EE 5

- The focus of Java EE 5 is ease of development
- How?
 - > EJBs as POJOs
 - > Make deployment descriptors meta data as annotations
 - > Use dependency injection to reduce or eliminate lookup code
 - > Better default behaviour and configuration
 - > Simplified container managed persistence (third time lucky)
- Developer work less, container work more

EJB 3.0

Simplification

Example Session Bean – 1

```
// EJB 2.1

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");
    }
}
```

Note: Home, Remote and/or Local interfaces not shown

Example Session Bean – 2

```
// EJB 2.1 (cont.)

public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbRemove() {}

public void setBenefitsDeduction(
    int empId, double deduction) {

    ...
    Connection conn = empDB.getConnection();
    ...
}
```

Example Deployment Descriptor

```
<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>
```

EJB 3.0 Approach

- Simplification of the EJB APIs
 - > Removal of need for EJBHomes and EJBObjects
 - > Removal of JNDI APIs from developer and client view
 - > Elimination of need for deployment descriptors
- Utilizes advantages of Java language metadata
 - > Leverages defaulting for typical cases
 - > Metadata designed so that the most common cases are the easiest to express

EJB 3.0 Approach

- More work is done by container, less by developer
- Inversion of contracts
 - > Bean specifies what it *needs* through metadata
 - > No longer written to unneeded container interfaces
 - > Container interpositions to provide *requested* services

Dependency Injection

- Resources a bean depends upon are injected when bean instance is constructed
- References to:
 - > EJBContext
 - > DataSources
 - > UserTransaction
 - > Environment entries
 - > EntityManager
 - > TimerService
 - > Other EJB beans
 - > ...

Dependency Injection

- Annotations
 - > @EJB
 - > References to EJB business interfaces
 - > References to Home interfaces (when accessing EJB 2.1 components)
 - > @Resource
 - > Almost everything else
 - > Number of annotations is simplified from EJB 3 specification early draft
- Injection can also be specified using deployment descriptor elements

Example Session Bean



```
// Same example, EJB 3.0
```

```
@Stateless
```

```
@TransactionManagement(BEAN)
```

```
public class PayrollBean  
    implements Payroll {
```

```
    @Resource DataSource empDB; //Using default name
```

```
    @Resource UserTransaction userTx;
```

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

```
    ...
```

Event Notification

- Need to implement a bunch of “ejb” methods
 - > `ejbCreate()`, `ejbActivate()`, `ejbRemove()`, etc
 - > Even if we don't use them
- Use annotations to define define only those events that we are interested in
 - > Removes boilerplate code
- Container defines the following events
 - > `@PostConstruct`
 - > `@PreDestroy`
 - > `@PostActivate`
 - > `@PrePassivate`

Lifecycle Example – EJB 2.1

```
public class AccountManagementBean
    extends SessionBean {

    private Socket sc = null;
    public void ejbCreate() {
        sc = new Socket(DEST_HOST, DEST_PORT);
        ...
    }
    public void ejbRemove() {
        sc.close();
    }

    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void setSessionContext(
        SessionContext sc) { }
    ...
}
```

Lifecycle Example – EJB 3

```
@Session public class AccountManagementBean
    implements AccountManagement {

    private Socket sc = null;

    @PreConstruct @PostActivate
    public void initSocket() {
        sc = new Socket(DEST_HOST, DEST_PORT);
        ...
    }

    @PreDestroy @PrePassivate
    public void closeSocket() {
        sc.close();
    }
    ...
}
```



EJB 3.0

Persistence

Persistence Goals of EJB 3.0

- Simplify entity bean programming model
- Support for light-weight domain modelling, including:
 - > Inheritance and polymorphism
- Complete query capabilities
- Support for object/relational mapping specification
- Make entity instances usable outside the EJB container
 - > Remove need for DTOs and similar antipatterns

Persistence Model in EJB 3.0

- Entities are simple Java classes
 - > Concrete classes—support use of new
 - > Getter/setter “property” methods or persistent instance variables
 - > No required bean interfaces
 - > No required callback interfaces
- Usable as “detached” objects in other application tiers
 - > No more need for DTOs

Persistence Model in EJB 3.0

- Entities are simple Java classes
 - > Persistent fields and relationships are annotated
 - > JavaBeans “getter/setter” methods can be used as well
- Usable as “detached” objects in other application tiers
 - > No need for additional class to implement DTO pattern

CMP Example – EJB 3.0

```
@Remote @Entity(access=FIELD) @Table(name = "customer")
public class Customer {
    @Id public int id;
    ...
    public String name;

    @Column(name="CREDIT") public int c_rating;

    @LOB public Image photo;

    @ManyToMany
        @JoinTable(table=@Table(name="CUST_PHONE")
            , joinColumns=@JoinColumn(name="CUST_ID")
            , inverseJoinColumns=@JoinColumn (name="PHON_ID"))
    public Collection<Phone> phones;
    ...
}
```

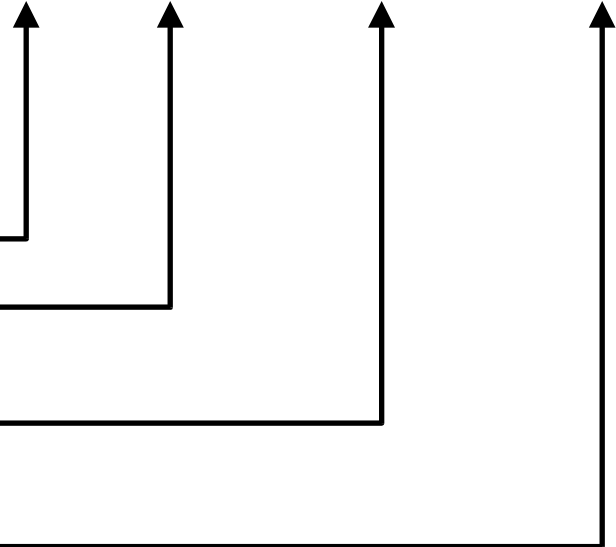


EJB 3.0 CMP – Simple Mapping



```

@Entity(access=FIELD)
public class Customer {
    @Id
    int id;
    String name;
    @Column(name="CREDIT")
    int c_rating;
    @Lob
    Image photo;
}
    
```



EJB 3.0 CMP – ManyToMany Mapping

```

@Entity(access=FIELD)
public class Customer {
    ...
    @ManyToMany
    @JoinTable(table=@Table(name="CUST_PHONE"),
        joinColumns=@JoinColumn(name="CUST_ID"),
        inverseJoinColumns=@JoinColumn(name="PHON_ID"))
    Collection<Phone> phones;
}

```



EJB Inheritance

```
@Entity @Table(name="EMP") @Inheritance(strategy=JOINED)
public abstract class Employee {
    @Id protected Integer empId;
    @Version protected Integer version;
    @ManyToOne protected Address address;
    ...
}
```

Entity may be specified as an abstract class

```
@Entity @Table(name="FT_EMP") @DiscriminatorValue("FT")
@PrimaryKeyJoinColumn(name="FT_EMPID")
public class FullTimeEmployee extends Employee {
    protected Integer salary;
    public Integer getSalary() { return salary; }
    ...
}
```

EJB 3.0 CMP – Inheritance Mapping

Single table:

ANIMAL				
ID	DISC	NAME	LEG_CNT	WING_SPAN

Joined:

ANIMAL	
ID	NAME

LAND_ANML	
ID	LEG_COUNT

AIR_ANML	
ID	WING_SPAN

Table per Class:

LAND_ANML		
ID	NAME	LEG_COUNT

AIR_ANML		
ID	NAME	WING_SPAN

Simplified Client View

- Session beans have plain Java language business interface
 - > No more EJB(Local)Home interface
 - > No more EJB(Local)Object interface
- Bean class implements interface
 - > Looks like normal Java class to bean developer
- Looks like normal Java language interface to client (POJI)

Client Example – EJB 2.1

```
Context ctx = new InitialContext();
```

```
ShoppingCartHome home = (ShoppingCartHome)  
    ctx.lookup("java:comp/env/ejb/myCart");
```

```
ShoppingCart cart = home.create();
```

```
Collection widgets = cart.startToShop("widgets");
```

Client Example – EJB 3.0

```
//Use resource injection to get a Session Bean  
@EJB ShoppingCart myCart;
```

...

```
Collection widgets = myCart.startToShop("widgets");
```

...



Entity Manager

```
@Stateless public ShoppingCartBean
    implements ShoppingCart {

    @PersistenceContext EntityManager entityManager;

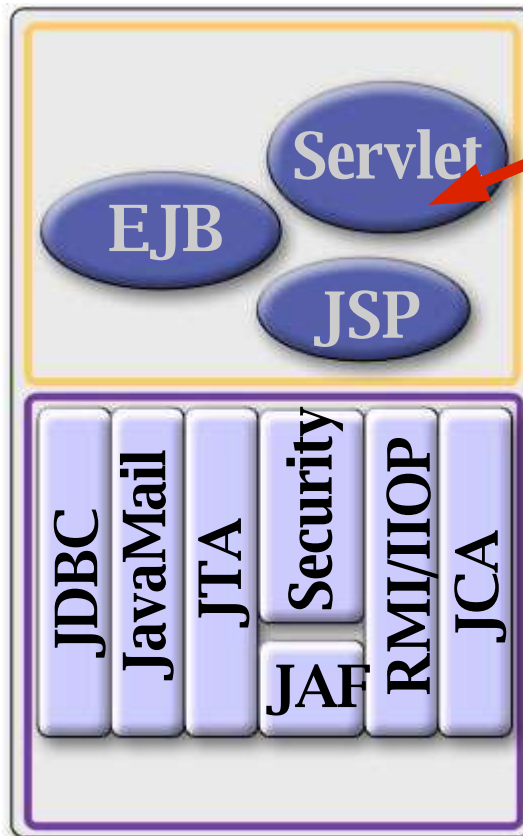
    public OrderLine createOrderLine(Product product
        , Order order) {
        OrderLine orderLine = new OrderLine(order, product);
        entityManager.persist(orderLine);
        return (orderLine);
    }

    public OrderLine updateOrderLine(OrderLine orderLine) {
        return (entityManager.merge(orderLine));
    }
}
```



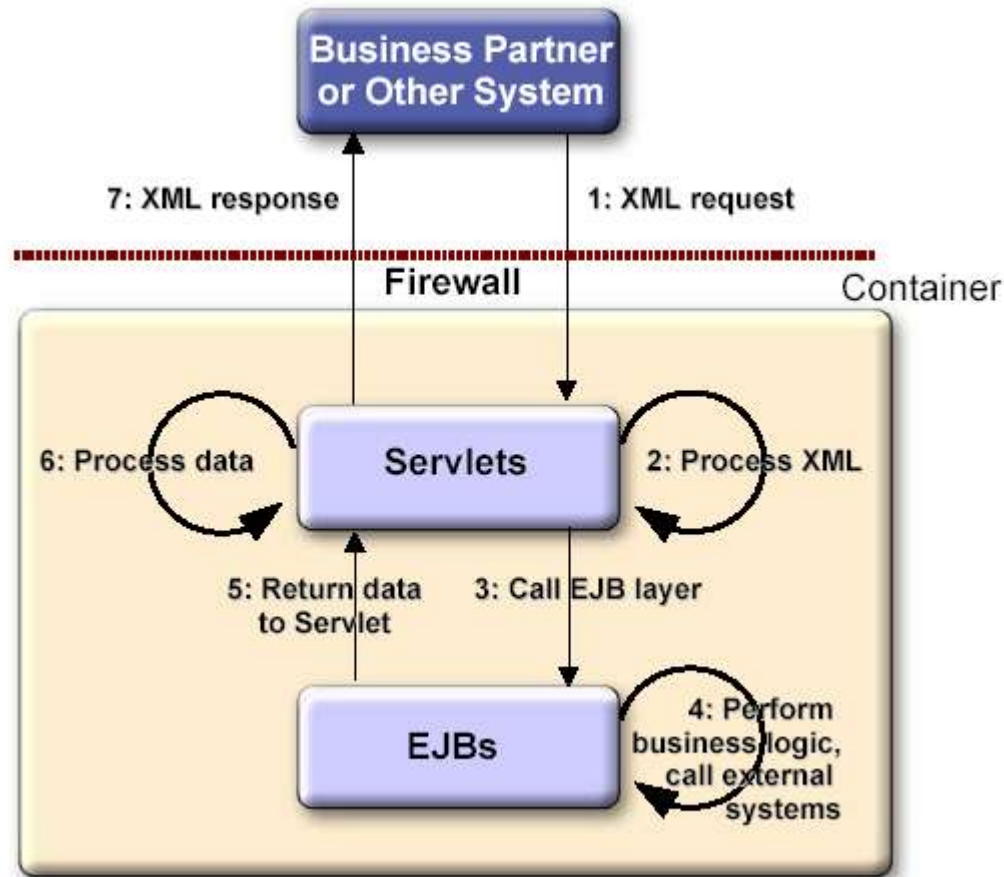
Web Services

Leveraging Existing Components



**Potential
web services**

JAX-WS Technology



Current Web Services Development

- Need to remember rules when developing Java Web Services
- Complicated APIs
 - > Gets complicated once you try to do more sophisticated things
 - > Serializers, handlers
- Need to leverage Tiger features
 - > Annotations to specify web service
 - > Future<?> for asynchronous web service

Simplification of Web Services

- JAX-RPC is a good starting point
 - > Relatively simple to develop web services
 - > Do not need to know XML!!!!
- Allow user focus on business logic
- Explicitly define the characteristics of the web service
 - > Declarative instead of programmatic
 - > Portable to different tools
 - > Do not hide these in the deployment descriptors

Hello World Web Service

@WebService ← Declaring class as a web service

```
public class HelloWorld {
```

@WebMethod ← Declaring a method as an operation

```
    public String sayHelloWorld() {  
        return ("Hello World!");  
    }  
}
```



A Slightly More Complicated Example

@WebServices

**@SOAPBinding(style=SOAPBinding.Style.DOCUMENT
, use=SOAPBinding.Style.LITERAL)**

```
public class EchoService {  
    @WebMethod  
    public String echo(@WebParam(name="myHeader"  
        , header=true, mode=WebParam.Mode.INOUT)  
        StringHolder header, String msg) {  
  
        String result = header.value + ", " + msg;  
        header.value = "got it";  
        return (result);  
    }  
}
```



Example adapted from BEA's JavaOne 2006 presentation

Summary

- Java EE 5 New Features
- EJB 3.0 Simplified API and EoD Feature
- EJB 3.0 Persistence Model
- Simplification of web services development
- Liberal use of annotations

Sun Developer Network

- Free resources for all developers
 - > Tools (Creator, Java Studio Enterprise Edition)
 - > Forums
 - > FAQs
 - > Early Access
 - > Newsletters
- Join Today!
 - > <http://developers.sun.com>



Get Connected

Sun Developer Network connects you to what you need, when you need it.

[Join Now >>](#)

Resource

- Enterprise Java Bean TM Specification
<http://www.jcp.org/en/jsr/detail?id=220>
- JavaTM Enterprise Edition 5 Specification
<http://www.jcp.org/en/jsr/detail?id=244>
- Sun Java Studio Creator
<http://developers.sun.com/prodtech/javatools/jscreator/index.jsp>
- Netbeans.org
- Sun Java Studio Enterprise
<http://developers.sun.com/prodtech/javatools/jsenterprise/index.html>



Thank You!

Lee Chuk Munn

Senior Developer Consultant
Sun Microsystems, Inc.