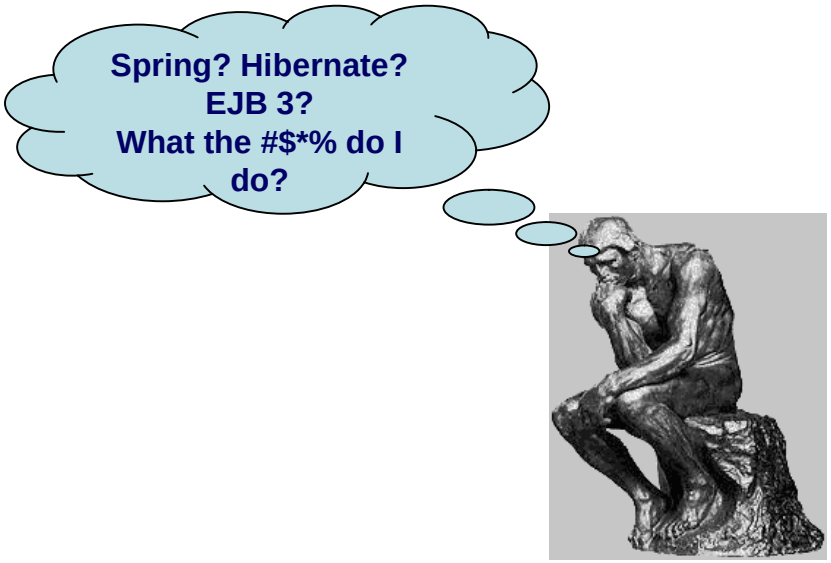
EJB 3, Spring and Hibernate

A Comparative Analysis



Reza Rahman

Author, *EJB 3 in Action*

Chief Software Architect, Tripod Technologies

Philadelphia Java User Group

September 2007

The Continuing Conundrum: What are the Cool Kids Wearing Today?

- **EJB 2.x, the architect's darling in the bubble years, is now officially as un-cool as grandpa in Speedos (and rightfully so!):**
  - Made you write a bunch of redundant artifacts.
  - Gator wresting with deployment descriptors.
  - Entity beans are half-baked and non-portable.
- **Enter Spring and Hibernate, the cool new kids on the block:**
  - POJO programming.
  - Give you 90+% of what EJB does.
  - And then some.
- **EJB 3 to the rescue? Does granddad know something the cool kids don't after all?**

EJB 3: EJB Reinvented

• EJB 3 is truly a different beast from EJB 2.x. It is an overhaul of the magnitude its pioneers would not have foreseen:

- EJB 3 embraces POJO programming through annotations.
- The verbose XML deployment descriptor has been made optional.
- The Entity Bean concept is no longer managed by the container.
- Instead, EJB 3 adopts JPA, an API based paradigm similar to Hibernate, TopLink and JDO.
- Object relational mapping and object queries have been completely defined instead of being left up to container vendors to sort out.
- EJB 3 makes heavy use of “intelligent defaulting” whenever possible. This is a similar idea to “convention over configuration” in the Rails world.

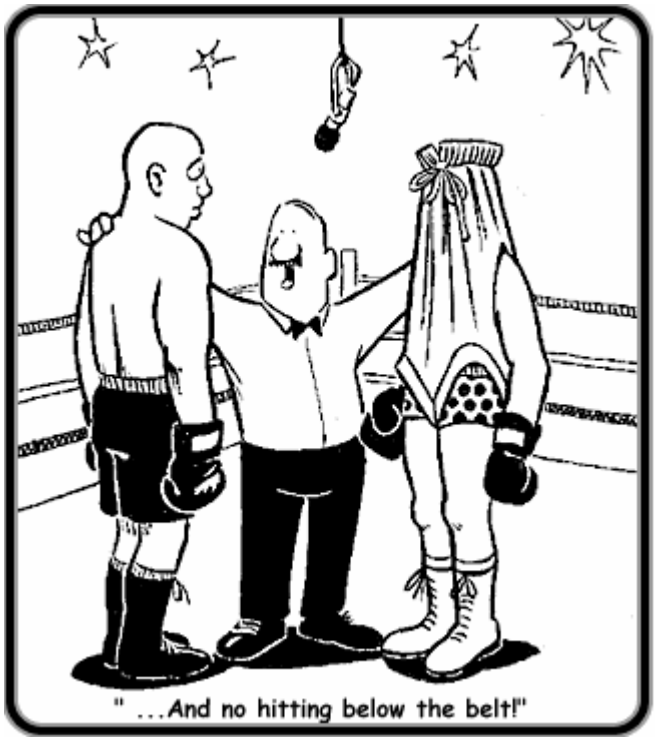
• These changes make EJB 3 a viable development choice on its own right. It just may be the easiest full stack enterprise platform to work with.

• However, the question remains as to why developers should opt for EJB 3 instead of simply sticking with Spring and Hibernate.

EJB 3, Spring and Hibernate: The Bottom Line

- **Use EJB 3 if:**
  - You like annotations and dislike XML configuration.
  - You prefer a tightly integrated solution stack that makes sensible default choices for you and keeps configuration to a bare minimum.
  - Your application is very stateful.
  - You use JSF and are considering using Seam.
  - Standardization is an important consideration.
- **Use Spring if:**
  - Your application requires fine-grained control at the container level.
  - Your application requires a lot of configuration beyond gluing together components and resources.
  - You need to build your own solution stack (such as with iBATIS, Quartz or Acegi).
  - You need advanced AOP features.
- **Use Hibernate if:**
  - You like the Hibernate API better than JPA.
  - You are very unlikely to switch persistence engines.

EJB 3 and Spring



EJB 3 and Spring Feature Sets

	EJB 3	Spring
Dependency Injection	Can inject anything in the container including EJBs, data sources, JMS resources, JPA resources and web services endpoints	Can inject almost anything including lists, maps, properties and JNDI resources
Transaction management	Works right out of the box, but only JTA is supported	Have to configure it to make it work, but supports a number of strategies including JTA, JDBC and Hibernate
Persistence	Tightly integrated through JPA	Framework support for JPA, JDO, Hibernate, JDBC, iBatis
State management	Robust support through Stateful Session Beans and Extended Persistence Context	Indirect support dependent on web container session management
Web services	Seamless support for JAX-WS 2.0	Poor direct support, best integration available is via configuring XFire for registered beans.
Messaging	Robust support out of the box through Message Driven Beans	Need to add configuration for each message listener. Listeners are not thread-safe.
Remoting	Integrated support through Session Bean remote interfaces. Supports distributed transactions and security.	Remoting support may be added via configuration. Remote transactions and security are not supported. However protocols other than RMI such as HTTP, Hessian and Burlap are supported.
AOP	Simple but limited support through interceptors.	Robust support through AspectJ.
Security	Integrated support for declarative and programmatic security through JAAS.	Must add and configure Acegi security. However, support beyond JAAS is possible through Acegi.
Scheduling	Simple scheduling possible through EJB Timer service.	Must configure Quartz for scheduling.
Clustering	Most containers have built-in support.	Complex third-party solutions available.

A Quick Look at Code: EJB 3

```
@Stateless
public class PlaceBidBean implements PlaceBid {
    @PersistenceContext
    private EntityManager entityManager;

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void addBid(Bid bid) {
        entityManager.persist(bid);
    }
}

@Remote
@WebService
public interface PlaceBid {
    void addBid(Bid bid);
}
```

Stateless Code

A Quick Look at Code: Spring Java Code

```
public class PlaceBidBean implements PlaceBid {
    private SessionFactory sessionFactory;

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Transactional(propagation=Propagation.REQUIRED)
    public void addBid(Bid bid) {
        sessionFactory.getCurrentSession().save(bid);
    }
}

public interface PlaceBid {
    void addBid(Bid bid);
}
```

Spring Stateless Code

A Quick Look at Code: Spring XML Configuration

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="hibernateProperties">
      <value>hibernate.dialect=org.hibernate.dialect.HSQLDialect</value>
    </property>
</bean>

<bean id="transactionManager"
      class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>

<tx:annotation-driven transaction-manager="transactionManager"/>

<bean id="placeBid" class="PlaceBidBean">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>

<bean class="org.springframework.remoting.rmi.RmiServiceExporter">
    <property name="serviceName" value="placeBid"/>
    <property name="service" ref="placeBid"/>
    <property name="serviceInterface" value="PlaceBid"/>
    <property name="registryPort" value="1199"/>
</bean>

<bean id="placeBidService" class="org.codehaus.xfire.spring.remoting.XFireExporter">
    <property name="serviceFactory" ref="xfire.serviceFactory"/>
    <property name="xfire" ref="xfire"/>
    <property name="serviceBean" ref="placeBid"/>
    <property name="serviceClass" value="PlaceBid"/>
</bean>

<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="urlMap">
        <map>
            <entry key="/PlaceBidService"><ref bean="placeBidService"/></entry>
        </map>
    </property>
</bean>
```

Spring XML Configuration

web.xml code

EJB 3 State Management

```
@Stateful
public class BidderAccountCreatorBean implements BidderAccountCreator {
    @PersistenceContext(type=PersistenceContextType.EXTENDED)
    private EntityManager entityManager;

    public void addLoginInfo(LoginInfo loginInfo) {
        entityManager.persist(loginInfo);
    }

    public void addBiographicalInfo(BiographicalInfo biographicalInfo) {
        entityManager.persist(biographicalInfo);
    }

    public void addBillingInfo(BillingInfo billingInfo) {
        entityManager.persist(billingInfo);
    }

    @Remove
    public void cancelAccountCreation() {
        entityManager.clear();
    }

    @Remove
    public void createAccount() {
        entityManager.flush();
    }
}
```

EJB 3 Stateful Code

Spring Dependency Injection Range

```
<bean id="placeBid" class="PlaceBidBean">
    <property name="bidDao" ref="bidDao"/>
    <property name="concurrencySensitivity" value="1"/>
    <property name="adminEmails">
        <props>
            <prop key="administrator">
                administrator@somecompany.org
            </prop>
            <prop key="support">
                support@somecompany.org
            </prop>
            <prop key="development">
                development@somecompany.org
            </prop>
        </props>
    </property>
    <property name="dataSource">
        <jee:jndi-lookup jndi-name="jdbc/ActionBazaarDB"/>
    </property>
</bean>
```

EJB 3 Dependency Injection Range

```
@Stateless
public class DIExampleBean implements DIExample {
    @Resource
    private SessionContext context;

    @PersistenceContext
    private EntityManager entityManager;

    @Resource(name="jdbc/TurtleDS")
    private DataSource dataSource;

    @EJB
    private PlaceBid placeBid;

    @WebServiceRef(wsdlLocation=
       "http://www.ripoffcreditprocessing.com/jaxws-webservice/BillingService?WSDL")
    private BillingService billingService;
}
```

Spring AOP through AspectJ

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Auditable {
    Boolean value() true;
}

@Aspect
public class AuditAspect {

    @Before("execution(public * *(..)) && @annotation(Auditable)")
    public void audit(JoinPoint joinPoint) {
         System.out.println("Entering: " + joinPoint);
         System.out.println("  with args: " + joinPoint.getArgs());
    }
}

public class PlaceBidBean implements PlaceBid {
    private SessionFactory sessionFactory;

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Auditable
    public void addBid(Bid bid) {
        sessionFactory.getCurrentSession().save(bid);
    }
}

<aop:aspectj-autoproxy />

<bean id="auditAspect" class="AuditAspect" />
<bean id="placeBid" class="PlaceBidBean">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

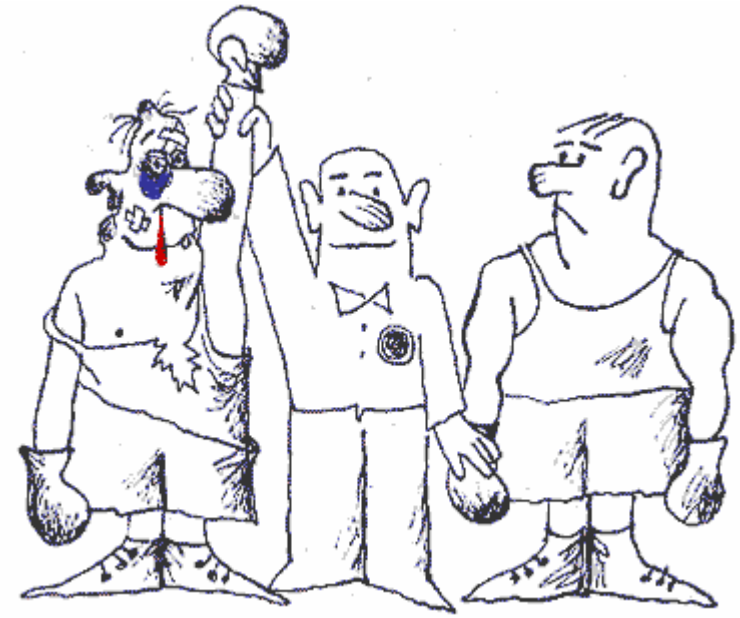
Lightweight EJB 3 AOP through Interceptors

```
public class AuditInterceptor {

    @AroundInvoke
    public Object audit(InvocationContext context) throws Exception {
         System.out.println("Entering: "
             + context.getMethod().getName());
         System.out.println("  with args: "
             + context.getParameters());
         return context.proceed();
    }
}

@Interceptors({AuditInterceptor.class})
@Stateless
public class PlaceBidBean implements PlaceBid {
    @PersistenceContext
    private EntityManager entityManager;

    public void addBid(Bid bid) {
        entityManager.persist(bid);
    }
}
```

Spring and EJB 3 Integration

- **Using EJB 3 in a Spring application:**
  - Very easy to do since Spring can be deployed inside a Java EE container.
  - Anything in JNDI can be injected into the Spring application context.
  - Especially easy using the **`jee`** schema: **`<jee:jndi-lookup/>`**, **`<jee:local-slsb/>`** and **`<jee:remote-slsb/>`** tags.
  - Spring provides excellent integrated support for JPA, very similar to Spring-Hibernate integration.
  - Spring natively supports EJB 3 features through the Pitchfork project.
- **Using Spring in an EJB 3 application:**
  - Insert Spring beans into EJB 3 using Interceptors. JBoss provides excellent support for doing this using the **`@Spring`** annotation and the JBoss Spring deployer. You can easily implement similar functionality in other Java EE containers yourself.

EJB 3 (JPA) and Hibernate



JPA and Hibernate Feature Sets

	JPA	Hibernate
ORM mapping	Supported through annotations and XML.	Supported through annotations and XML. Annotations are compatible with EJB 3 standard.
Mapping relations	Allows mapping of one-to-one, one-to-many, many-to-one and many-to-many relations	Allows mapping of one-to-one, one-to-many, many-to-one and many-to-many relations
Mapping complex types	Mapping complex types such as collections, composite keys, embedded classes and dates possible	Mapping complex types such as collections, composite keys, embedded classes and dates possible
Primary key generation	Various useful strategies for generating primary keys.	Various useful strategies for generating primary keys.
Inheritance	Full support for inheritance	Full support for inheritance
Object queries	Full support of object queries through JPQL	Full support of object queries through HQL
Native queries	Robust support for native queries	Robust support for native queries
API syntax	API defined by EJB 3 standard. Slightly smaller API	Proprietary API. Relatively larger API
Query syntax	More strict and SQL like	Relatively more flexible, but may be confusing for an SQL developer
Query API	Query API supports simple primary key based lookup and JQPL	Query API supports primary query based lookup and HQL. In addition, Query-by-Criteria/Query-by-Example is supported

A Quick Look at Code: The JPA EntityManager

```
public class ItemManager {
    private EntityManagerFactory factory;
    private EntityManager entityManager;

    public ItemManager() {
        factory = Persistence.createEntityManagerFactory("ActionBazaar");
        entityManager = factory.createEntityManager();
    }

    public void addItem(Item item) {
        EntityTransaction transaction = entityManager.getTransaction();
        entityManager.persist(item);
        transaction.commit();
    }

    public void updateItem(Item item) {
        EntityTransaction transaction = entityManager.getTransaction();
        entityManager.merge(item);
        transaction.commit();
    }

    public void deleteItem(Item item) {
        EntityTransaction transaction = entityManager.getTransaction();
        entityManager.remove(entityManager.merge(item));
        transaction.commit();
    }

    public Item getItemByName(String name) {
        Query query = entityManager.createQuery(
            "SELECT i FROM Item i WHERE i.name = :itemName");
        query.setParameter("itemName", name);
        return (Item)query.getSingleResult();
    }

    public List getItems() {
        return entityManager.createQuery("SELECT i FROM Item i").getResultList();
    }

    public void close() {
        entityManager.close();
        factory.close();
    }
}
```

JPA Code

A Quick Look at Code: The Hibernate Session

```
public class ItemManager {
    private SessionFactory factory;
    private Session session;

    public ItemManager() {
        factory = new Configuration().configure().buildSessionFactory();
        session = factory.getCurrentSession();
    }

    public void addItem(Item item) {
        Transaction transaction = session.beginTransaction();
        session.save(item);
        transaction().commit();
    }

    public void updateItem(Item item) {
        Transaction transaction = session.beginTransaction();
        session.update(item);
        transaction.commit();
    }

    public void deleteItem(Item item) {
        Transaction transaction = session.beginTransaction();
        session.delete(item);
        transaction.commit();
    }

    public Item getItemByName(String name) {
        Item item;
        Transaction transaction = session.beginTransaction();
        Query query = session.createQuery("from Item i where i.name = :itemName");
        query.setParameter("itemName", name);
        item = (Item) query. uniqueResult();
        transaction.commit();
        return item;
    }

    public List getItems() {
        List items;
        Transaction transaction = session.beginTransaction();
        items = session.createQuery("from Item").list();
        transaction.commit();
        return items;
    }

    public void close() {
        session.close();
        factory.close();
    }
}
```

Hibernate Code

Annotated Entity Example (Hibernate or EJB 3)

```
@Entity
@Table(name="ITEMS")
public class Item implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="ITEM_ID", nullable=false)
    public Long itemId;

    @Column(name="ITEM_NAME")
    public String title;

    @Column(name="BID_END_DATE")
    public Timestamp bidEndDate;

    @Column(name="BID_START_DATE")
    public Timestamp bidStartDate;

    @Column(name="INITIAL_PRICE")
    public Double initialPrice;

    @OneToMany(mappedBy="item", cascade=CascadeType.ALL)
    public Set<Bid> bids;
}
```

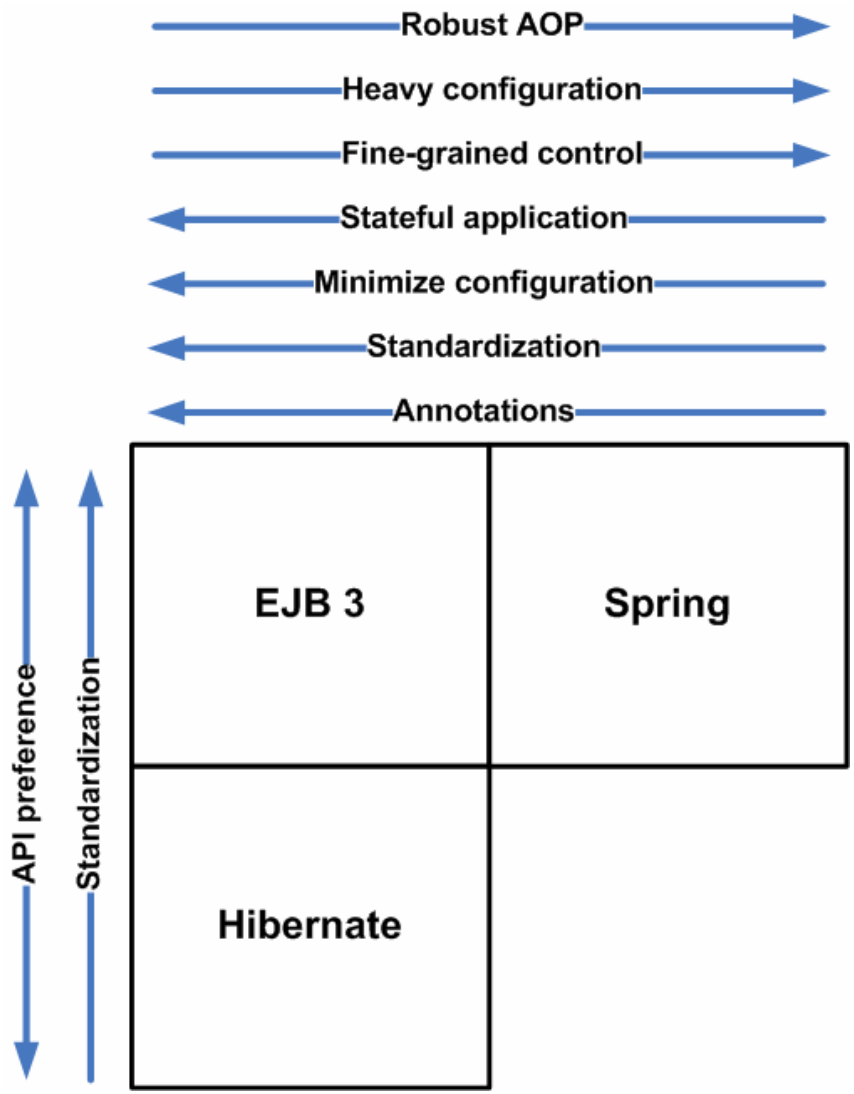
Entity Code

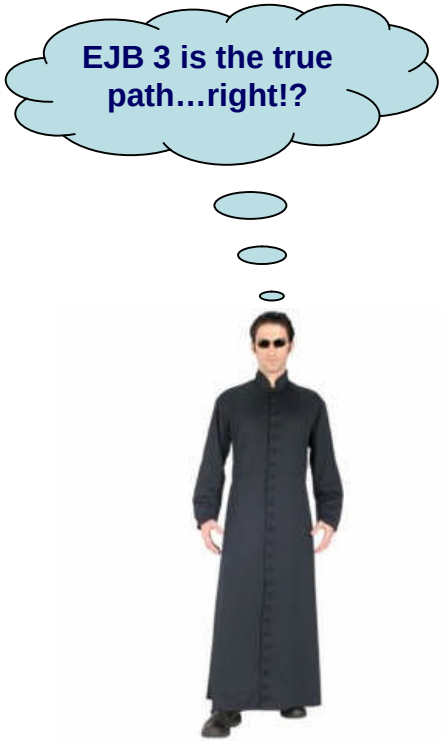
Query-by-Criteria/Query-by-Example

```
List items = session.createCriteria(Item.class)
    .add(Restrictions.like("name", "%Useless Junk%"))
    .add(Restrictions.between("initialPrice", cheapest, cheapo))
    .list();
```

```
Item junk = new Item();
junk.setBidStartDate(tomorrow);
junk.setBidEndDate(dayAfterTomorrow);
List results = session.createCriteria(Item.class)
    .add(Example.create(junk))
    .list();
```

The Matrix: EJB3, Spring and Hibernate

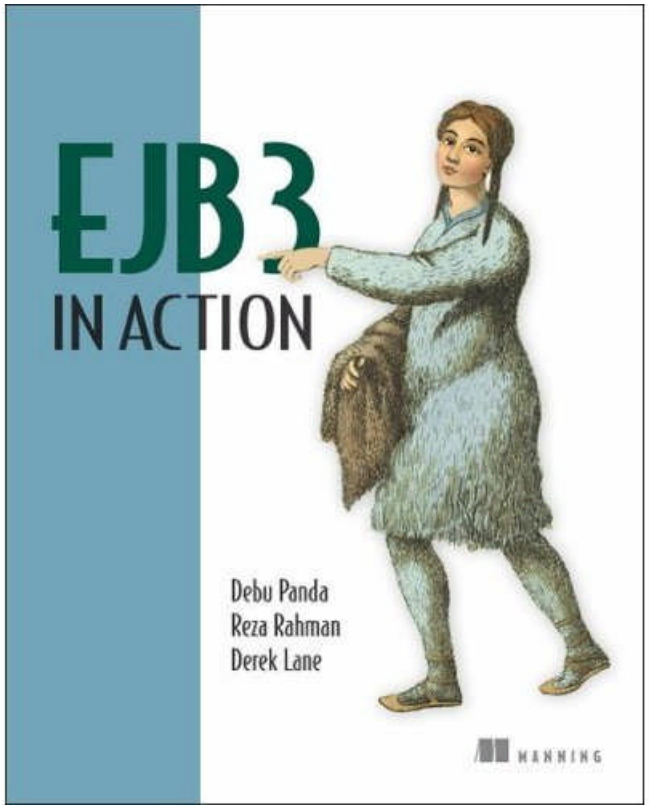




References

• **POJO Application Frameworks: Spring Vs. EJB 3.0**, Michael Juntao Yuan, http://www.onjava.com/pub/a/onjava/2005/06/29/spring-ejb3.html.

• **Make the Right Decision with Our Side-by-Side Comparison of Spring and EJB 3.0**, Rod Coffin, http://www.devx.com/Java/Article/32314/0/page/1.



Shameless plug alert!