Ease of Development in the Java™ Platform, Enterprise Edition 5

Roberto Chinnici
Senior Staff Engineer
Sun Microsystems, Inc.

roberto.chinnici@sun.com
Contents

- Why the focus on Ease of Development?
- The challenge
- The solution
- Selected examples from Java™ Platform, Enterprise Edition 5 ("Java™ EE 5")
- Are we done?
Caveat

- All new Java™ API go through the Java Community Process
- 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!
Why Ease of Development?

- People get used to the most amazing things
- Still, we have to admit too much is too much
- For the longest time, we said that better tools were the answer
- Isn't it better to have great tools and an easy to understand technology?
You Know You Have a Problem When...

- A beginner Enterprise JavaBeans™ (EJB) programmer runs into PortableRemoteObject.narrow on day one
- Long JNDI names have to be kept in sync in three places – and what's with the “comp/env”?
- The HelloWorld web service takes 80 lines and 3 deployment descriptors to work
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
Developer Segmentation

App Complexity

Enterprise

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

Corporate

Content

Colorado Software Summit: October 23 – 28, 2005
© Copyright 2005, Sun Microsystems, Inc.
Why Is It So Challenging?

- 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
Tools at Our Disposal

- New language features
  - Annotations, generics
- Better understanding of how components in the platform are used
  - e.g. No scriptlets, no remoting for entity beans
- Guideline: “make simple things easy, complex things possible”
- 80-20 rule
Sometimes Tools Bite Back

- **Good**
  
  List<E>.iterator() returns Iterator<E>
  
  List<E>.addAll(Collection<? Extends E>)

- **Bad**
  
  Class Enum<E extends Enum<E>>
  
  Set<Entry<? extends String, ? extends Object>> props = map.entrySet();


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 when needed
Benefits to Beginners

- Java™ EE programming closer to the SE model
  - More intuitive programming model
- Less to be learnt on day one
- Hide creation/lookup of resources
  - What you need (e.g. database) is “right there”
- Reduce references to “hard” APIs
- Code is more readable
Benefits to Intermediate/Experts

- Build on existing knowledge
- Less boilerplate code
- Focus on what is important
- Fewer mandated interfaces = more freedom
- More uniformity across the platform
- Easier to create unit tests
Big Ticket Items

- Simplified component model
  - EJB 3.0 session beans
- New unified object persistence
  - EJB 3.0 entities
- New web services stack:
  - JAX-WS 2.0, JAXB 2.0, JSR-181, JSR-109.next
- JavaServer™ Faces 1.2 and JSTL 1.2 now standard in the platform
Examples

EJB 2.1 vs 3.0
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");
    }
}
EJB 2.1 3.0 Session Bean

@Stateless
public class PayrollBean
implements javax.ejb.SessionBean {

@Resource SessionContext ctx;
@Resource 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");
}
EJB 2.1 Session Bean (Continued)

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

public void setBenefitsDeduction(int empId, double deduction) {
    ...
    Connection conn = empDB.getConnection();
    ...
}
...
EJB 2.1 3.0 Session Bean (Continued)

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

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

```
<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 2.13.0 Session Bean (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>
```
Dependency Injection

- Applies to all container-managed objects
- Class-, field- or method-based
- Marked using:
  - @Resource, @EJB, @WebServiceRef
- Common elements:
  - name, type, description, mappedName
- Plus type-specific ones:
  - authenticationType, beanName, wsdlLocation, ...
Resolving Dependencies – Before

```java
try {
    Context initial = new InitialContext();

    Object objref = initial.lookup
                     ("java:comp/env/ejb/Request");
    RequestHome home = (RequestHome)
                        PortableRemoteObject.narrow(objref,
                                          RequestHome.class);
    Request request = home.create();

    // ... use the request bean ...

} catch (NamingException ex) {
    // ...
} catch (CreateException ex) {
    // ...
}
```
Resolving Dependencies – Now

@EJB Request request;
Behind the Scenes

- Resource type inferred from the field type
- JNDI name defaulted
- Automatic resolution of EJB link based on type
- Injection happens at construction time
- Exceptions prevent the component from being constructed
- If any of these assumptions are invalid, developer can take control
Why Three Kinds of Injection?

- Good reasons for all of them
- Class: declare a resource but don't inject it
  ```
  @Resource(type=EntityManager.class)
  class MyManagedBean { ... }
  ```
- Method: need to initialize the resource
  ```
  @Resource void setDB(DataSource mydb) { ... }
  ```
- Field: reasonable default
  ```
  @Resource(name="jms/airline/queue")
  Queue myQueue;
  ```
Lifecycle Events – On Demand

@Stateless
public class PayrollBean {

@Resource SessionContext ctx;
@Resource DataSource empDB;

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

@PostConstruct
void initialize() { ... }

}
EJB 2.1 Transaction Attributes

```xml
<container-transaction>
  <method>
    <ejb-name>PayrollBean</ejb-name>
    <method-intf>Local</method-intf>
    <method-name>setBenefitsDeduction</method-name>
    <method-params>
      <method-param>int</method-param>
      <method-param>double</method-param>
    </method-params>
  </method>
  <trans-attribute>Required</trans-attribute>
</container-transaction>
```
EJB 3.0 Transaction Attributes

@TransactionAttribute(MANDATORY)
public void setBenefitsDeduction(int empId,
        double deduction) { ... }
EJB 3.0 Entities

- Concrete classes, no required interfaces
- Support inheritance
- Usable in “detached” mode, replacing the Data Transfer Object (DTO) antipattern
- Created via an EntityManager
  - “Home” for all entities in a persistence unit
  - Also factory for Query objects
- Usable on Java™ EE and Java™ SE
Quick Check

- We got rid of:
  - Deployment descriptors
  - JNDI lookups
  - PortableRemoteObject.narrow
  - Several try/catch blocks
  - Boilerplate ejbXYZ() methods
  - DTOs
  - Session vs. entity bean dilemma
What Happened to DDs?

- Deployment descriptors optional
- Format similar to previous version
  - Ease of migration
- New elements for injection, etc.
- “Full” bit for descriptors that capture all the information for a module
  - Think of it as “compiling annotations”
Descriptors Are Optional

- Still useful for assembler, deployer
- In general, descriptors override annotations
- Made distinction between overridable and non-overridable annotations
  - e.g. Cannot make a session bean into an entity
- Some descriptors were not brought forward
  - e.g. jaxrpc-mapping-info.xml
Web Services
Example – JAX-RPC 1.1 Web Service

```java
public interface HelloService extends Remote {
    public String sayHello(String name) throws RemoteException;
}

public class HelloServiceBean implements SessionBean {
    public String sayHello(String name) {
        return "Hello " + name + " from HelloServiceBean";
    }
    public HelloServiceBean() {}
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext sc) {}
}
```
Example – JAX-WS 2.0 Web Service

```java
public interface HelloService extends Remote {
    public String sayHello(String name) throws RemoteException;
}

@WebService @Stateless
public class HelloServiceBean implements SessionBean {
    public String sayHello(String name) {
        return "Hello "+ name + " from HelloServiceBean";
    }
    public HelloServiceBean() {}
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext sc) {}
}
```
Example – Descriptors

JAX-RPC 1.1

```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>
    <port-component xmlns:wsdl-port_ns='urn:HelloService/wsdl'>
      <port-component-name>HelloService</port-component-name>
      <wsdl-port>wsdl-port_ns:HelloServiceSEIPort</wsdl-port>
      <service-endpoint-interface>endpoint.HelloServiceSEI</service-endpoint-interface>
      <service-impl-bean>
        <servlet-link>WSServlet_HelloService</servlet-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

JAX-WS 2.0

```xml
<?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>
```
Simplified!

- No need to define an interface
- No more RMI-like markers
- No more generated stubs, ties
- Annotations used to customize WSDL mapping
- No mapping file needed
  - Data binding done by JAXB 2.0
- No webservices.xml descriptor needed
  - @WebService annotation declares an endpoint
Example – Web Service

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

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

}
```
Web Service Packaging

- WSDL is optional
  - If missing, generated at deployment time
- No need for a descriptor to point to the WSDL:
  ```java
  @WebService(wsdlLocation="META-INF/wsdl/my.wsdl")
  public class PayrollService { ... }
  ```
- Within an EAR file, web service clients and endpoints are tied together by qualified name
  - No more port-component/ejb-link elements
Example – JAX-RPC 1.1 Client

```java
try {
    Context ic = new InitialContext();
    MyHelloService myHelloService = (MyHelloService) ic.lookup("java:comp/env/service/MyJAXRPCHello");
    HelloIF helloPort = myHelloService.getHelloIFPort();

    // ... use the service ...

} catch (NamingException ex) {
    // ...
} catch (RemoteException ex) {
    // ...
}
```
Example – JAX-WS 2.0 Client

- Inject a service reference:

```java
@WebServiceRef
MyHelloService helloService;

HelloIF helloPort = myHelloService.getHelloIFPort();
```

- Or inject a port reference directly:

```java
@WebServiceRef(MyHelloService.class)
HelloIF helloPort;
```
Example – JAX-RPC 1.1 Client (SE)

```java
try {
    ServiceFactory factory = ServiceFactory.newInstance();
    MyHelloService myHelloService =
        factory.loadService(MyHelloService.class);
    HelloIF helloPort = myHelloService.getHelloIFPort();

    // ... use the service ...

} catch (ServiceException ex) {
    // ...
} catch (RemoteException ex) {
    // ...
}
```
Example – JAX-WS 2.0 Client (SE)

```java
MyHelloService myHelloService = new MyHelloService();
HelloIF helloPort = myHelloService.getHelloIFPort();

// ... use the service ...
```
JAXB 2.0

- Supports all of XML Schema
  - No need to drop down to DOM
  - Can use complex, industry-standard schemas directly
- Class-first model in addition to schema-first
- Integrated with JAX-WS 2.0
- Generated code is portable
- Structural mapping captured in annotations
Example – JAXB 2.0

Source code

```java
@XmlElement
public String symbol;

@XmlAttribute
int getQuantity() {...} void setQuantity() {...}
```
RESTful Web Service Client

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

AcmeFooRequest request = new AcmeFooRequest();

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

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

// ... process the response ...
Examples

Other Technologies
Injection Available Everywhere

- EJB container
  - Beans, interceptors, handlers
- Web container
  - Servlets, servlet filters, event listeners
  - Tag handlers, tag library event listeners
  - Managed beans
  - JAX-RPC/JAX-WS endpoints, handlers
- Client
  - Main class, login callback handler
JavaServer™ Faces 1.2

- UI component model using familiar patterns
- Tool-friendly
- Logic mostly in managed beans
  - Plain JavaBeans™ objects
  - Instantiated on-demand
  - With injection, even simpler to write!
- Next version will use annotations to declare beans
Example – Managed Bean Config

```xml
<managed-bean>
<!-- Customer Bean created on demand -->
<managed-bean-name>logonBean</managed-bean-name>
<managed-bean-class>
  mypackage.LogonBean
</managed-bean-class>
<managed-bean-scope>request</managed-bean-scope>
<managed-property>
  <property-name>creditLimit</property-name>
  <value>${initParam.defaultCreditLimit}</value>
</managed-property>
</managed-bean>
```
JDBC 4.0

- Not part of Java™ EE 5, but available when running on Mustang (Java™ SE 6)

- Introducing **DataSet:**
  - Type-safe view of the result of a SQL Query
  - Used with user-defined data classes and query objects
  - Can be connected or disconnected
    - Akin to ResultSet, CachedRowSet
  - Big improvement in ease of use
Example – JDBC 3.0

String query = "SELECT NAME, PRICE FROM COFFEES";
ResultSet rs = stmt.executeQuery(query);
while (rs.next()) {
    String s = rs.getString("NAME");
    float n = rs.getFloat("PRICE");
    System.out.println(s + "  " + n);
}
public class CoffeeInfo {
    String name;
    float price;
}

public class CoffeeQuery {
    @Query("SELECT NAME, PRICE FROM COFFEES")
    DataSet<CoffeeInfo> getAllInfo();
}

CoffeeQuery q = con.createQueryObject(CoffeeQuery.class);
DataSet<CoffeeInfo> data = q.getAllInfo();
for (info : data) {
    System.out.println(info.name + " " + info.price);
}
Scripting in the Java™ Platform

- JavaScript support in Java™ SE 6
- Optional web container API
- Many languages potentially available:
  - JavaScript
  - Groovy, BeanShell
  - Jython, JRuby
  - PHP
- Mix scripting and Java™ components
Example – JavaScript “servlet”

// hello script

response.setContentType("text/html");
writer = response.getWriter();
writer.println("<html><head><title>Hello</title></head>
" + "<body>Hello!</body></html>");
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