

# The JavaEE 6 Platform

A yellow starburst graphic with the word "Done!" written inside in black text, positioned to the right of the "6" in the "JavaEE 6" text.

Alexis Moussine-Pouchkine  
Antonio Goncalves



# Overall Presentation Goal

Focus on the new features of Java EE 6  
Write a web application

*Better if you know Java EE 5*

This is no science fiction



Java EE 6 and GlassFish v3  
shipped final releases on  
December 10<sup>th</sup> 2009

# Agenda

- Overview of EE 6 and GlassFish v3
- Dive into some specs & demos
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - Bean Validation 1.0
  - JAX-RS 1.1
  - CDI 1.0
- Summary

# Antonio Goncalves

- Freelance software architect
- Former BEA consultant
- Author (Java EE 5 and Java EE 6)
- JCP expert member
- Co-leader of the Paris JUG
- Les Cast Codeurs podcast
- Java Champion



# Alexis Moussine-Pouchkine

- GlassFish Ambassador at Sun Microsystems
- 10-year Sun and AppServer veteran
- Speaker at multiple conferences
- Your advocate for anything GlassFish



# Agenda

- Overview of EE 6 and GlassFish v3
- Dive into some specs & demos
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - Bean Validation 1.0
  - JAX-RS 1.1
  - CDI 1.0
- Summary

# Let's write a Java EE 6 app!

- Demos throughout the talks
- Nothing better than a few demos to understand better (when they work)
- A simple application to create Books and CDs

# DEMO

The application we will be writing

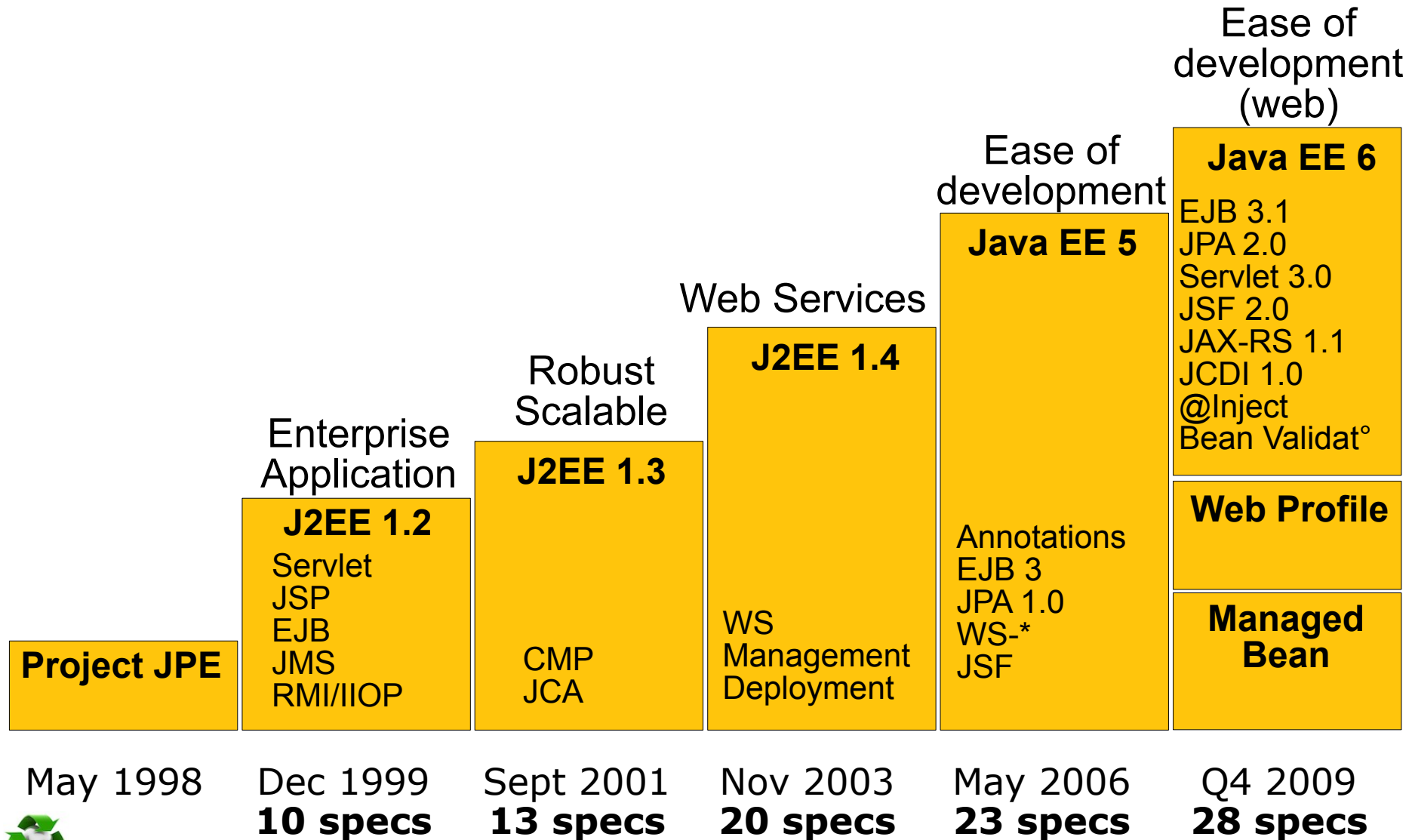
# GlassFish v3

<http://glassfish.org>



- The Reference Impl. (RI) for Java EE 6
  - Home of Metro, Grizzly, Jersey, Mojarra and other sub-projects
- Yet, production-quality and open source
  - Fast growing number of production deployments
- Modular (OSGi) and Extensible (HK2)
- Developer friendly
- Final today!

# A brief history



# Zomming in Java EE 6

## Web

JSF	2.0
Servlet	3.0
JSP	2.2
EL	2.2
JSTL	1.2

## Enterprise

EJB	3.1
JAF	1.1
JavaMail	1.4
JCA	1.6
JMS	1.1
JPA	2.0
JTA	1.1

## Web Services

JAX-RPC	1.1
JAXM	1.0
JAX-WS	1.1
JAXR	1.0
Web Services	1.3
WS Metadata	2.0

## Management, Security, Common

CDI (JSR 299)	1.0
@Inject (JSR 330)	1.0
Bean Validation	1.0
Interceptors	1.1
Managed Beans	1.0
JACC	1.3
Java EE Application Deployment	1.2
Java EE Management	1.1
JASPIC	1.0
Debugging Support	1.0

## + Java SE 6

JAX-WS	2.2
JAXB	2.2
JDBC	4.0
JNDI	1.5
SAAJ	1.3
Common Annotations	1.1
RMI	
Java IDL	
JMX	
JAAS	
JAXP	
StAX	

...

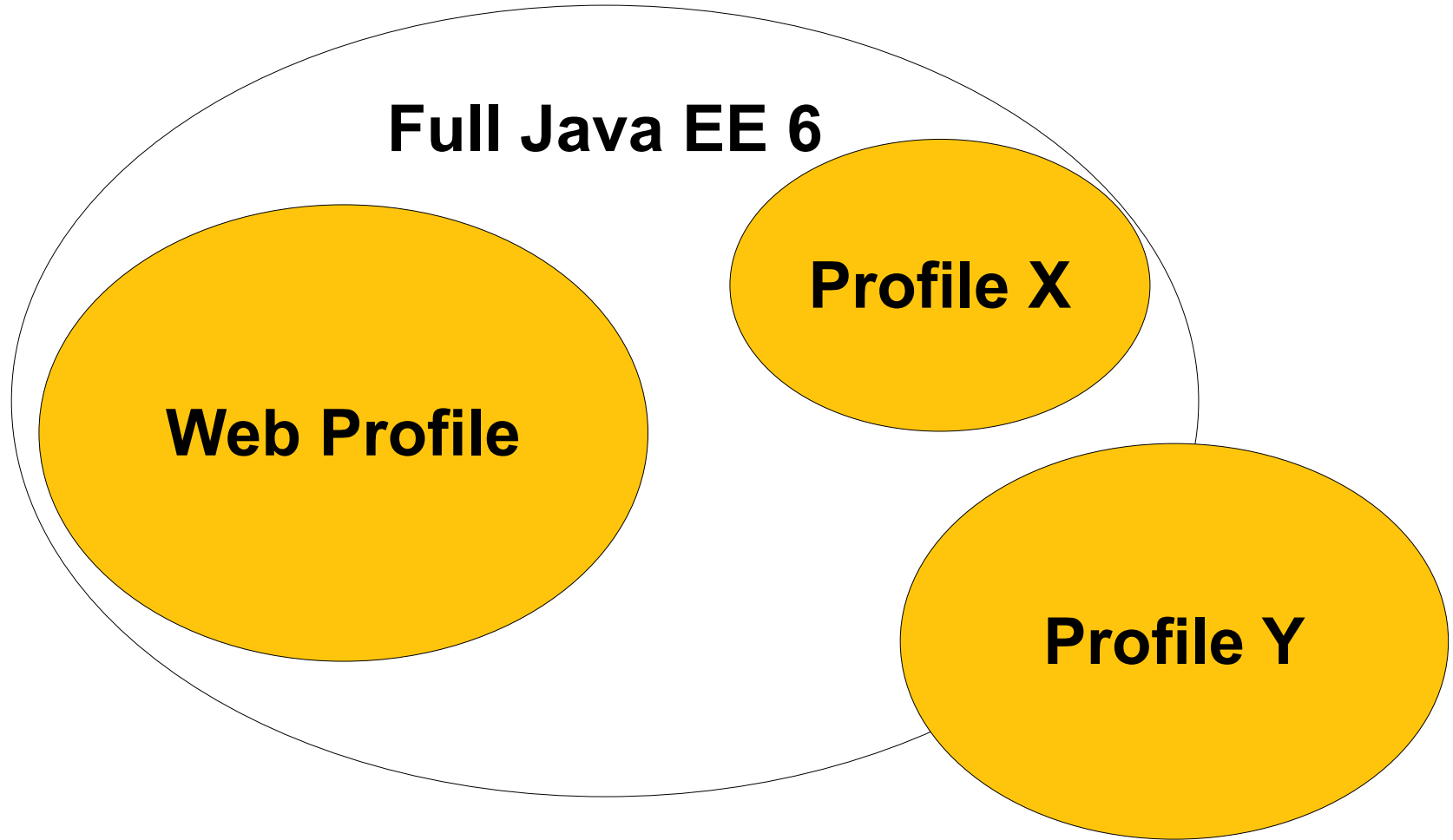
# New concepts

- Pruning
- Profiles
- EJB Lite
- Portable JNDI names
- Managed Beans

# Pruning (Soon less specs)

- Marks specifications optional in next version
- Pruned in Java EE 6
  - Entity CMP 2.x
  - JAX-RPC
  - JAX-R
  - JSR 88 (Java EE Application Deployment)
- Might disappear from Java EE 7

# Profiles



# Web Profile

- Subset of full platform
- For web development
  - Packages in a war
- Separate specification
- Evolves at its own pace
- Others will come
  - Minimal (Servlet/JSP)
  - Portal....

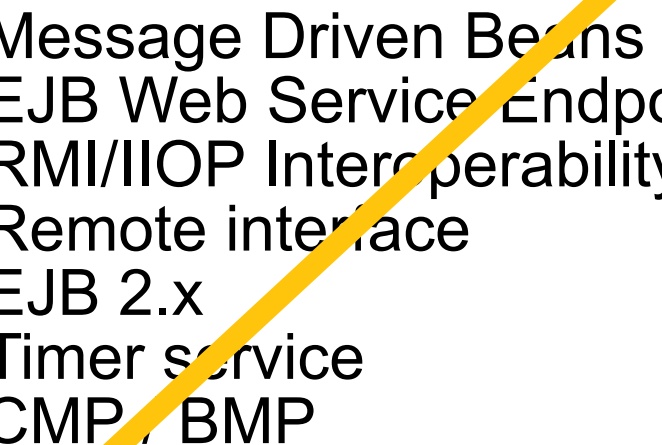
JSF	2.0
Servlet	3.0
JSP	2.2
EL	2.2
JSTL	1.2
<b>EJB Lite</b>	3.1
Managed Beans	1.0
Interceptors	1.1
JTA	1.1
JPA	2.0
Bean Validation	1.0
CDI	1.0
@Inject	1.0

# EJB Lite

- Subset of the EJB 3.1 API
- Used in Web profile
- Packaged in a war

Local Session Bean  
Injection  
CMT / BMT  
Interceptors  
Security

Message Driven Beans  
EJB Web Service Endpoint  
RMI/IIOP Interoperability  
Remote interface  
EJB 2.x  
Timer service  
CMP / BMP



# Portable JNDI names

- **Client inside a container (use DI)**

```
@EJB Hello h;
```

- **Client outside a container**

```
Context ctx = new InitialContext();  
Hello h = (Hello) ctx.lookup("xyz");
```

- **Portable JNDI name is specified**

```
java:global/env/foo/HelloEJB
```

# Portable JNDI names

- `java:comp`
  - Names in this namespace are per-component
- `java:module`
  - Names shared by all components in a module
- `java:app`
  - Shared in all modules of an application (.ear)
- `java:global`
  - Names shared by all applications deployed in an application server

# Managed Beans 1.0

- Separate spec shipped with Java EE 6
- Container-managed POJOs
- Support a small set of basic services
  - Injection (`@Resource...`)
  - Life-cycle (`@PostConstruct`, `@PreDestroy`)
  - Interceptor (`@Interceptor`, `@AroundInvoke`)
- Lightweight component model



# Managed Beans 1.0

```
@javax.annotation.ManagedBean
```

```
public class MyPojo {
```

```
    @Resource
```

```
    private Datasource ds;
```

```
    @PostConstruct
```

```
    private void init() {
```

```
        ....
```

```
    }
```

```
@Interceptors (LoggingInterceptor.class)
```

```
    public void myMethod() {...}
```

```
}
```

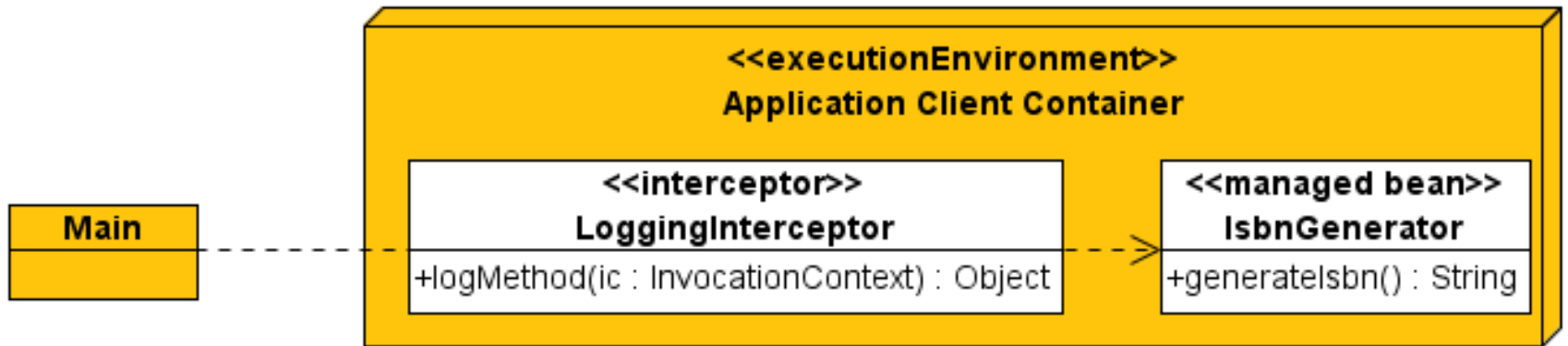
*JSR 250*

*Commons annotations*

# Managed Beans 1.0

- Everything becomes a Managed Bean with extra services
- An EJB is a Managed Bean with :
  - Transaction support
  - Security
  - Thread safety
- A REST service is a Managed Bean with :
  - HTTP support
- ...

# Demo : Managed Bean



# DEMO 01

Write a simple Managed Bean with  
Lifecycle callback annotations and an interceptor

# Agenda

- Overview of EE 6 and GlassFish v3
- Dive into some specs & demos
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - Bean Validation 1.0
  - JAX-RS 1.1
  - CDI 1.0
- Summary



# JPA 2.0

- Evolves separately from EJB now
  - JSR 317
- Richer mappings
- Richer JPQL
- Pessimistic Locking
- Criteria API
- Cache API

# Richer mapping

- Collection of embeddables and basic types
  - Not just collection of JPA entities
- Multiple levels of embeddables
- More flexible support for Maps
  - Keys, values can be : entities, embeddables or basic types
  - Support for ternary relationships
- More relationship mapping options
  - Unidirectional 1-many foreign key mappings
  - 1-1, many-1/1-many join table mappings

# Collections of Embeddable Types

```
@Embeddable public class BookReference {  
    String title;  
    Float price;  
    String description;  
    String isbn;  
    Integer nbOfPage;  
    ...  
}
```

```
@Entity public class ListOfGreatBooks {  
    @ElementCollection  
    protected Set<BookReference> javaBooks;  
    ...  
}
```

# Multiple levels of Embedding

```
@Embeddable public class BookReference {  
    @Embedded Author author;  
    ...  
}
```

```
@Entity public class Book {  
    @Id Long id;  
    String title;  
    BookReference theBook;  
    ...  
}
```

# Embeddable with Relationships

```
@Embeddable public class BookReference {  
    @Embedded Author author;  
    @OneToMany Set<Publisher> publishers;  
    ...  
}
```

```
@Entity public class Book {  
    @Id Long id;  
    String title;  
    BookReference theBook;  
    ...  
}
```

# Maps

```
@Entity public class BookStore {  
    @Id Integer storeID;  
    ...  
    @ElementCollection  
    Map<Book, Integer> inventory;  
    ...  
}
```

```
@Entity public class Book {  
    @Id Long id;  
    String title  
}
```

# Richer JPQL

- Added entity type to support non-polymorphic queries
- Allow joins in subquery `FROM` clause
- Added new reserved words
  - `ABS`, `BOTH`, `CONCAT`, `ELSE`, `END`,  
`ESCAPE`, `LEADING`, `LENGTH`, `LOCATE`,  
`SET`, `SIZE`, `SQRT`, `SUBSTRING`,  
`TRAILING`

# Standard properties

- In `persistence.xml` :
  - `javax.persistence.jdbc.driver`
  - `javax.persistence.jdbc.url`
  - `javax.persistence.jdbc.user`
  - `javax.persistence.jdbc.password`

# Locking Enhancements

- JPA 1.0 only supports optimist locking

```
public enum LockModeType {  
    OPTIMISTIC,  
    OPTIMISTIC_FORCE_INCREMENT,  
    PESSIMISTIC_READ,  
    PESSIMISTIC_WRITE,  
    PESSIMISTIC_FORCE_INCREMENT,  
    NONE,  
    READ,  
    WRITE  
}
```

- Methods added to `EntityManager`, `Query/TypedQuery`, **and** `lockMode` attribute for `NamedQuery` annotation

# Criteria API

- Strongly typed criteria API
- Object-based query definition objects, rather than string-based
- Operates on the metamodel
  - Abstract view of managed classes
  - `EntityManager.getMetamodel()`
- Each entity `X` has a metamodel class `X_`
- `CriteriaQuery` as a query graph

# Criteria API

```
EntityManager em = ...;  
CriteriaBuilder cb = em.getCriteriaBuilder();  
CriteriaQuery<Book> query =  
    cb.createQuery(Book.class);  
  
Root<Book> book = query.from(Book.class);  
  
query.select(book)  
    .where(cb.equal(book.get("description"), ""));
```

```
SELECT b  
FROM Book b  
WHERE b.description IS EMPTY
```

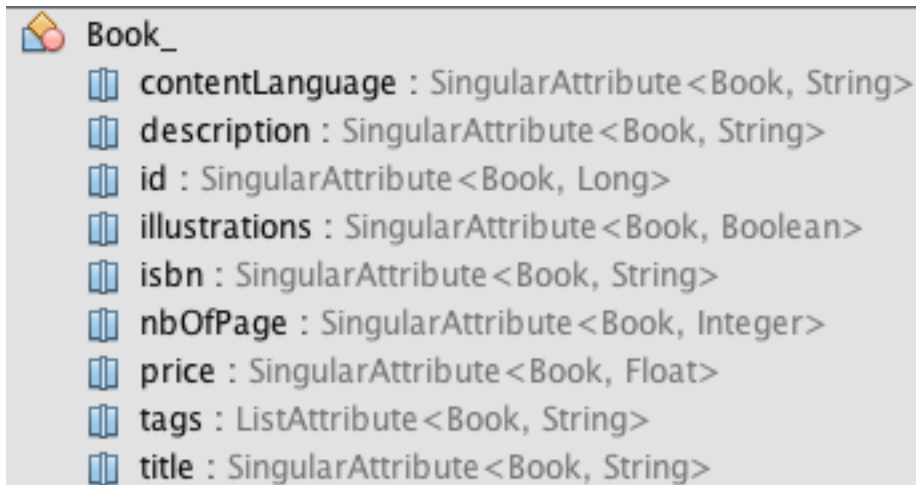
# Criteria API

## Type-safe

```
EntityManager em = ...;  
CriteriaBuilder cb = em.getCriteriaBuilder();  
CriteriaQuery<Book> query =  
    cb.createQuery(Book.class);
```

```
Root<Book> book = query.from(Book.class);
```

```
query.select(book)  
    .where(cb.isEmpty(order.get(Book_.description))));
```



Book\_

- contentLanguage : SingularAttribute<Book, String>
- description : SingularAttribute<Book, String>
- id : SingularAttribute<Book, Long>
- illustrations : SingularAttribute<Book, Boolean>
- isbn : SingularAttribute<Book, String>
- nbOfPage : SingularAttribute<Book, Integer>
- price : SingularAttribute<Book, Float>
- tags : ListAttribute<Book, String>
- title : SingularAttribute<Book, String>

Statically generated  
JPA 2.0 MetaModel



# Criteria API

## Joins and builder pattern

```
EntityManager em = ...;
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Customer> query =
    cb.createQuery(Customer.class);

Root<Customer> customer = query.from(Customer.class);

Join<Customer, Order> order =
    customer.join(Customer_.orders, JoinType.LEFT);

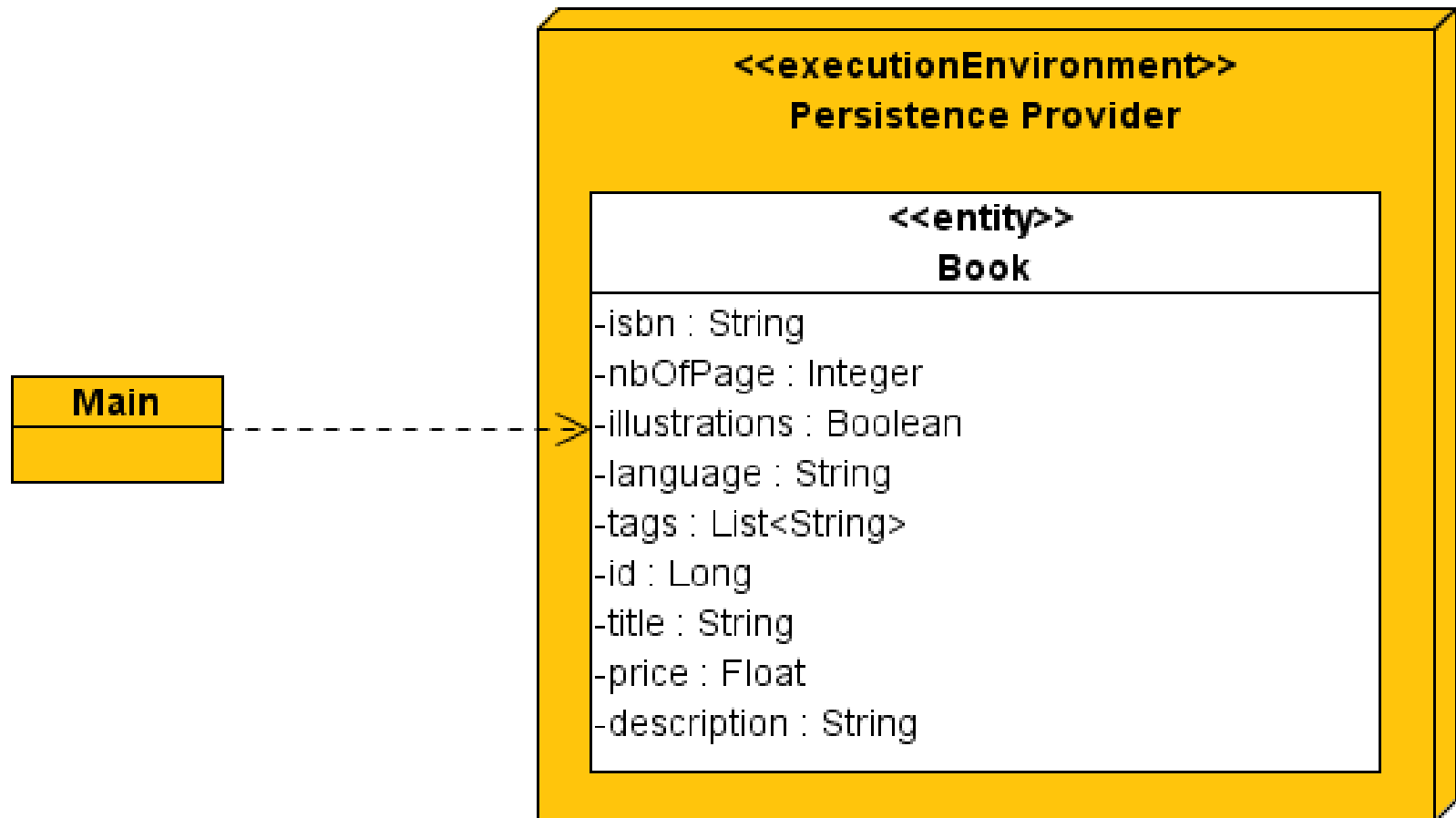
query.select(customer)
    .where(cb.equal(customer.get(Customer_.status), 1))
    .orderBy(...)
    .distinct(true)
    .groupBy(...);
```

```
SELECT c
FROM Customer c LEFT JOIN c.orders o
WHERE c.status = 1
```

# Caching

- Supports the use of a second-level cache
- Cache API
  - `contain(Class, PK)`
  - `evict(Class, PK), evict(Class)`
  - `evictAll()`
- `@Cacheable` annotation on entities

# Demo : Book Entity



# DEMO 02

Write a domain layer with JPA

# And more...

- Persistent order by
- `detach()` on EM and DETACH cascade
- Orphan removal functionality
  - `@OneToMany(orphanRemoval=true)`
- BeanValidation (JSR 303) integration
  - On `prePersist`, `preUpdate`, `preRemove`
  - Simply apply constraints to your `@Entity`
- Second-level cache API
  - `contain(Class, PK)`, `evict(Class, PK)`, ...
  - `@Cacheable` annotation on entities
- ...

# Agenda

- Overview of EE 6 and GlassFish v3
- Dive into some specs & demos
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - Bean Validation 1.0
  - JAX-RS 1.1
  - CDI 1.0
- Summary



# Servlets

- Probably most used Java EE technology
  - Most people now use servlets indirectly
- The servlet technology had little to no changes in Java EE 5
  - No annotation, no POJO, required `web.xml`
  - Still room for improvement
- Various Servlet 3.0 implementations
  - GlassFish v3 is the RI since Servlet 2.5
  - Tomcat 7, JBoss 6, Jetty 8, ...

# Servlet 3.0

- Ease of development
- Pluggability
- Asynchronous support

# Ease of development

- Annotations based programming model
  - `@WebServlet`
  - `@WebFilter`
  - `@WebListener`
  - `@WebInitParam`
- **Optional** `web.xml`
- **Better defaults and CoC**

# A servlet 2.5 example

```
public class MyServlet extends HttpServlet {
    public void doGet (HttpServletRequest req,
                      HttpServletResponse res) {
        ....
    }
}
```

## Deployment descriptor (web.xml)

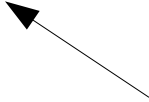
```
<web-app>
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>samples.MyServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/MyApp</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

# A servlet 3.0 example

```
@WebServlet(urlPatterns={"/MyApp"})  
public class MyServlet extends HttpServlet {  
  
    public void doGet (HttpServletRequest req,  
                      HttpServletResponse res) {  
  
        ....  
  
    }  
}
```

*web.xml is optional*



- Same for `@WebFilter`  
and `@WebListener`

# Pluggability

- Enable use of frameworks without configuration in `web.xml`
- Fragment the `web.xml` to allow frameworks to be self-contained in their own jar
- `/META-INF/resources` in any JAR to serve resources (applies to libraries)
- Dynamic container extension framework using `ServletContainerInitializer`
  - Simple JAR library manipulates `ServletContext` at startup

# Pluggability

## Web Fragments

- Fragments are similar to `web.xml`
- `<web-fragment>` instead of `<web-app>`
  - Declare their own servlets, listeners and filters
- Annotations and web fragments are merged following a configurable order
- JARs need to be placed in `WEB-INF/lib`
- and use `/META-INF/web-fragment.xml`
- Overridden by main `web.xml`
  - See `<metadata-complete>`,  
`<absolute-ordering>`

# Asynchronous support

- Servlets block waiting for a response
- Now they can start an asynchronous task...
- ...then, use a container callback...
- ...that invokes the servlet to end processing
- Better scalability
- New APIs for ServletRequest and Response
- Does not require NIO

# ChatServlet (1/2)

```
import javax.servlet.*;

Queue<AsyncContext> usersQueue;
BlockingQueue<String> messageQueue;

@WebServlet(urlPatterns = {"/chat"}, asyncSupported = true)
public class ChatServlet extends HttpServlet {

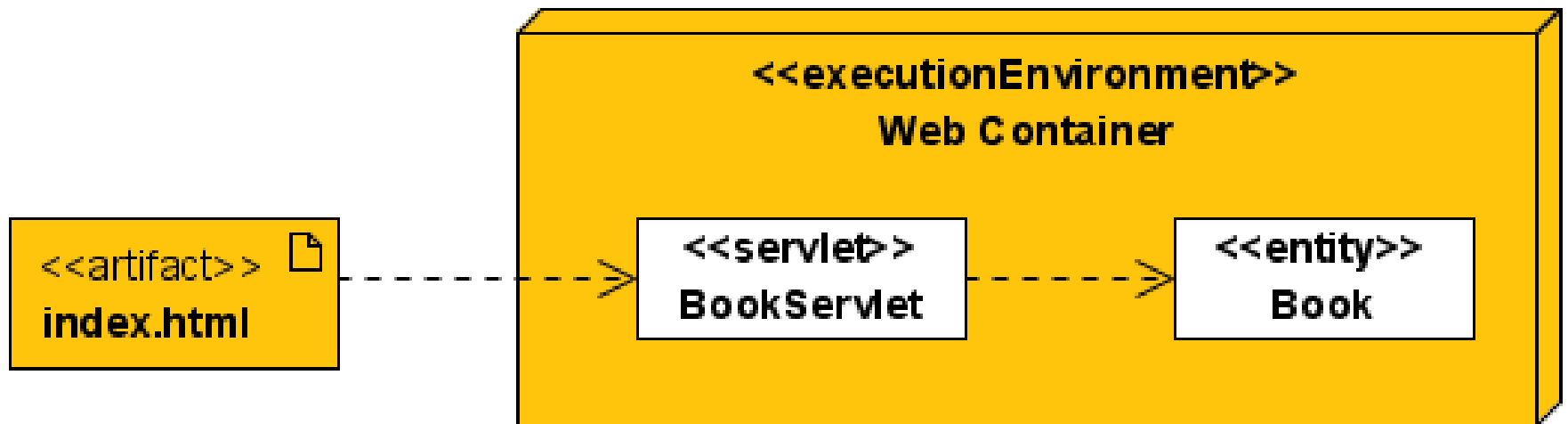
    doGet(...) {
        AsyncContext ac = req.startAsync();
        ac.setTimeout(10 * 60 * 1000);
        ac.addListener(new AsyncListener() {
            public void onComplete(AsyncEvent event) {
                usersQueue.remove(ac);
            } // deal with more events ...
        });
        usersQueue.add(ac);
    }
    ...
}
```

# ChatServlet (2/2)

```
doPost(...) {
    // also deal with system (login) messages
    String message = req.getParameter("message");
    messageQueue.put(message);
}

init(...) {
    while (true) {
        String message = messageQueue.take();
        for (AsyncContext ac : usersQueue) {
            PrintWriter acWriter =
                ac.getResponse().getWriter();
            acWriter.println(message);
        }
    }
}
}
```

# Demo : Add a Servlet



# DEMO 03

Add a servlet on top of domain layer

# And more...

- Asynchronous API
  - suspend/resume (Comet-like)
- Configuration API
  - Add and configure Servlet, Filters, Listeners
  - Add security constraints
  - Using `ServletContext` API
- File upload (similar to Apache File Upload)
- Configure cookie session name
- Security with `@ServletSecurity`

# Agenda

- Overview of EE 6 and GlassFish v3
- Dive into some specs & demos
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - Bean Validation 1.0
  - JAX-RS 1.1
  - CDI 1.0
- Summary



# EJB 3.1

- Interceptors
- Optional Local Interfaces
- Singleton
- Asynchronous calls
- Packaging in a war
- Cron-based Timer Service
- Embeddable Container
- EJB Lite

# Interceptors 1.1

- Address cross-cutting concerns in Java EE
- Were part of the EJB 3.0 spec
- Now a separate spec shipped with EJB 3.1
- Can be used in EJBs...
- ... as well as ManagedBeans
- `@AroundInvoke`
- `@AroundTimeout` for EJB timers

# Interceptors 1.1

- `@InterceptorBinding` annotation
- Or `<interceptor-binding>` XML element

```
<interceptor-binding>  
  <target-name>*Service</target-name>  
  <interceptor-class>MyIC</interceptor-class>  
  <interceptor-class>MyIC2</interceptor-class>  
  <method-name>create*</method-name>  
</interceptor-binding>
```

# EJB Optional Local Interface

- `@Local`, `@Remote`
- Interfaces are not always needed
  - Only for local interfaces
  - Remote interfaces are now optional !

**@Stateless**

```
public class HelloBean {  
  
    public String sayHello() {  
        return "Hello Devovx";  
    }  
}
```

# Asynchronous calls

- How to have asynchronous call in EJBs ?
  - JMS is more about sending messages
  - Threads and EJB's don't integrate well
- `@Asynchronous`
  - Applicable to any EJB type
  - Best effort, no delivery guarantee
- **Method returns** `void` or `Future<T>`
  - `java.util.concurrent` package
  - `javax.ejb.AsyncResult` helper class :  
`return new AsyncResult<int>(result)`

# Asynchronous calls

```
@Stateless
public class OrderBean {

    public void createOrder() {
        Order order = persistOrder();
        sendEmail(order); // fire and forget
    }

    public Order persistOrder() {...}

    @Asynchronous
    public void sendEmail(Order order) {...}
}
```

# Packaging in a war

`foo.ear`

`lib/foo_common.jar`

`com/acme/Foo.class`

`foo_web.war`

`WEB-INF/web.xml`

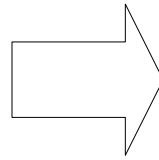
`WEB-INF/classes`

`com/acme/FooServlet.class`

`foo_ejb.jar`

`com/acme/FooEJB.class`

`com/acme/FooEJBLocal.class`



`foo.war`

`WEB-INF/classes`

`com/acme/Foo.class`

`com/acme/FooServlet.class`

`com/acme/FooEJB.class`

# Timer Service

- Programmatic and Calendar based scheduling
  - « Last day of the month »
  - « Every five minutes on Monday and Friday »
- Cron-like syntax
  - second [0..59], minute[0..59], hour[0..23]...
  - dayOfMonth[1..31]
  - dayOfWeek[0..7] or [sun, mon, tue..]
  - Month[0..12] or [jan,feb..]

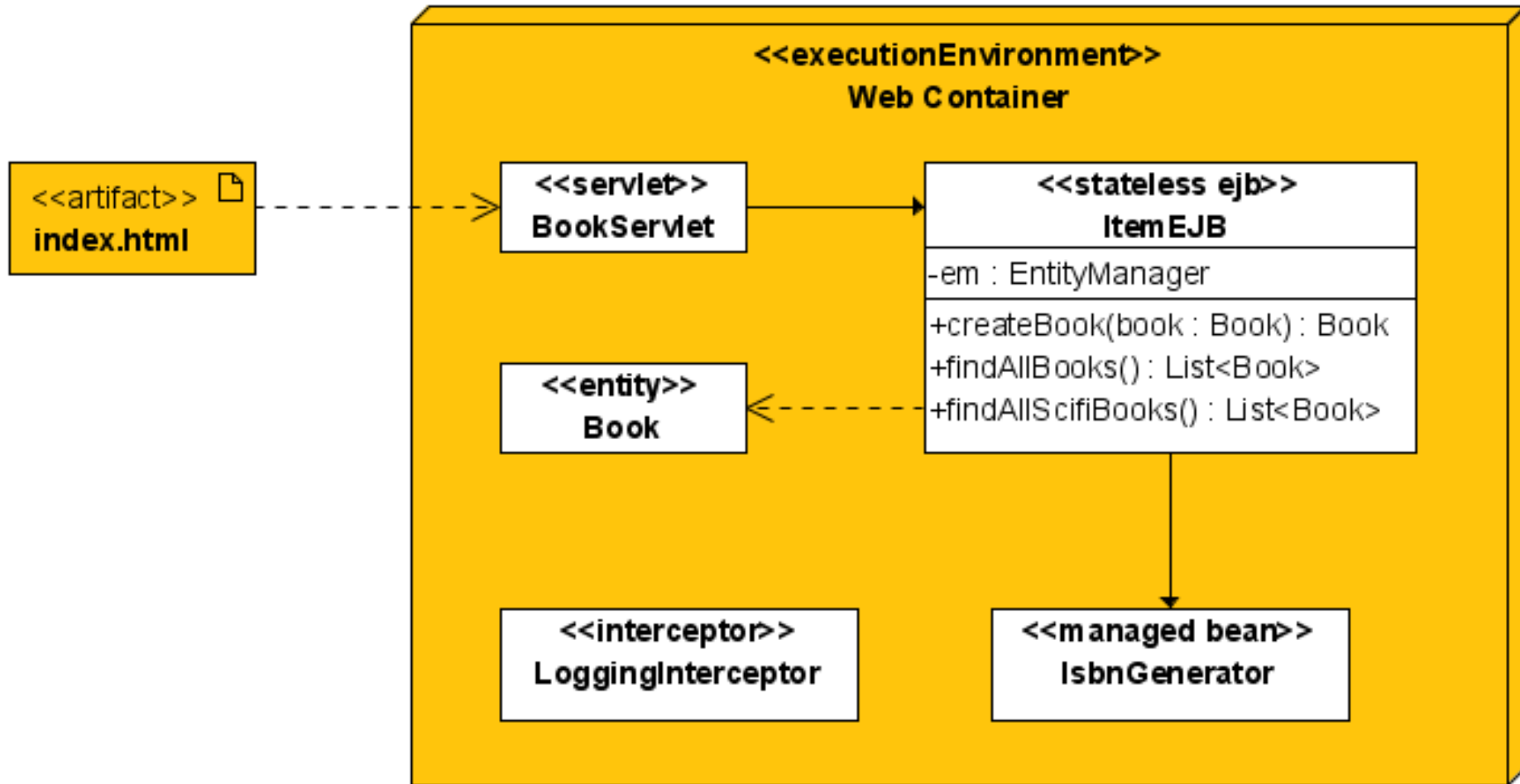
# Timer Service

```
@Stateless
public class WakeUpBean {

    @Schedule (dayOfWeek="Mon-Fri", hour="9")
    void wakeUp() {
        ...
    }
}
```

Deploy (potentially in a WAR file) is all you need  
No container config required

# Demo : add an EJB stateless



# DEMO 04

Add an EJB between the servlet and the entity

# Singleton

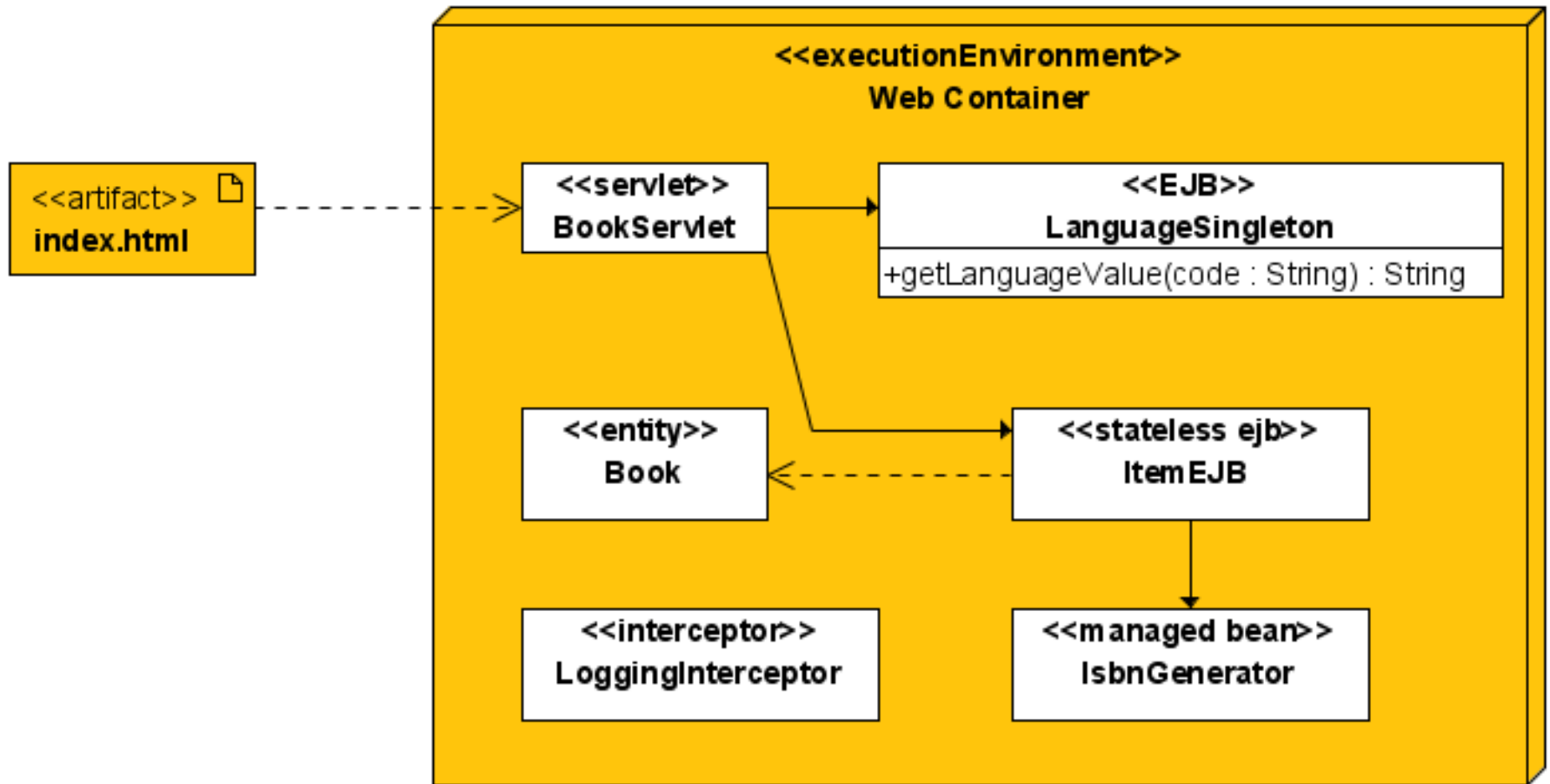
- New component
  - No/local/remote interface
- Follows the Singleton pattern
  - One single EJB per application per JVM
- Used to share state in the entire application
  - State not preserved after container shutdown
- Added concurrency management
  - Default is single-threaded
  - `@ConcurrencyManagement`

# Singleton

**@Singleton**

```
public class CachingBean {  
  
    private Map cache;  
  
    @PostConstruct void init() {  
        cache = ...;  
    }  
  
    public Map getCache() {  
        return cache;  
    }  
  
    public void addToCache(Object key, Object val) {  
        cache.put(key, val);  
    }  
}
```

# Demo : add a Singleton EJB

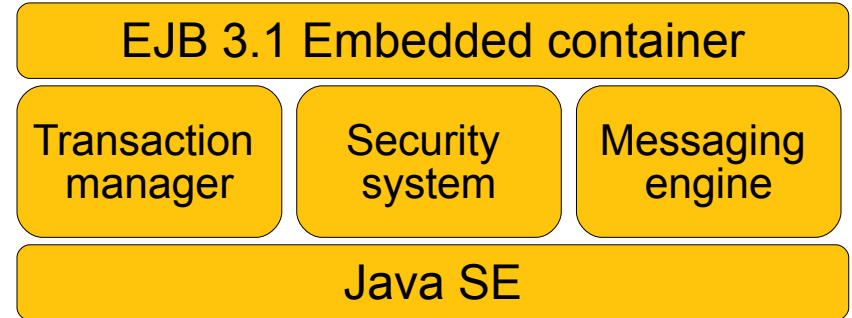


# DEMO 05

Add a Singleton to cache language codes

# Embeddable Container

- API allowing to :
  - Initialize a container
  - Get container ctx
  - ...



- Can run in any Java SE environment
  - Batch processing
  - Simplifies testing
  - Just a jar file in your classpath



# Embeddable Container

...

```
public static void main(String[] args) {  
  
    EJBContainer container =  
        EJBContainer.createEJBContainer() ;  
  
    Context context = container.getContext() ;  
  
    Hello h = (Hello)  
        context.lookup("java:global/classes/HelloEJB") ;  
  
    h.sayHello ;  
  
    container.close() ;  
}
```

...

# DEMO 06

Testing the EJB

# And more...

- Singletons can be chained
- Non persistent timer
- `@StatefulTimeout`
- ...

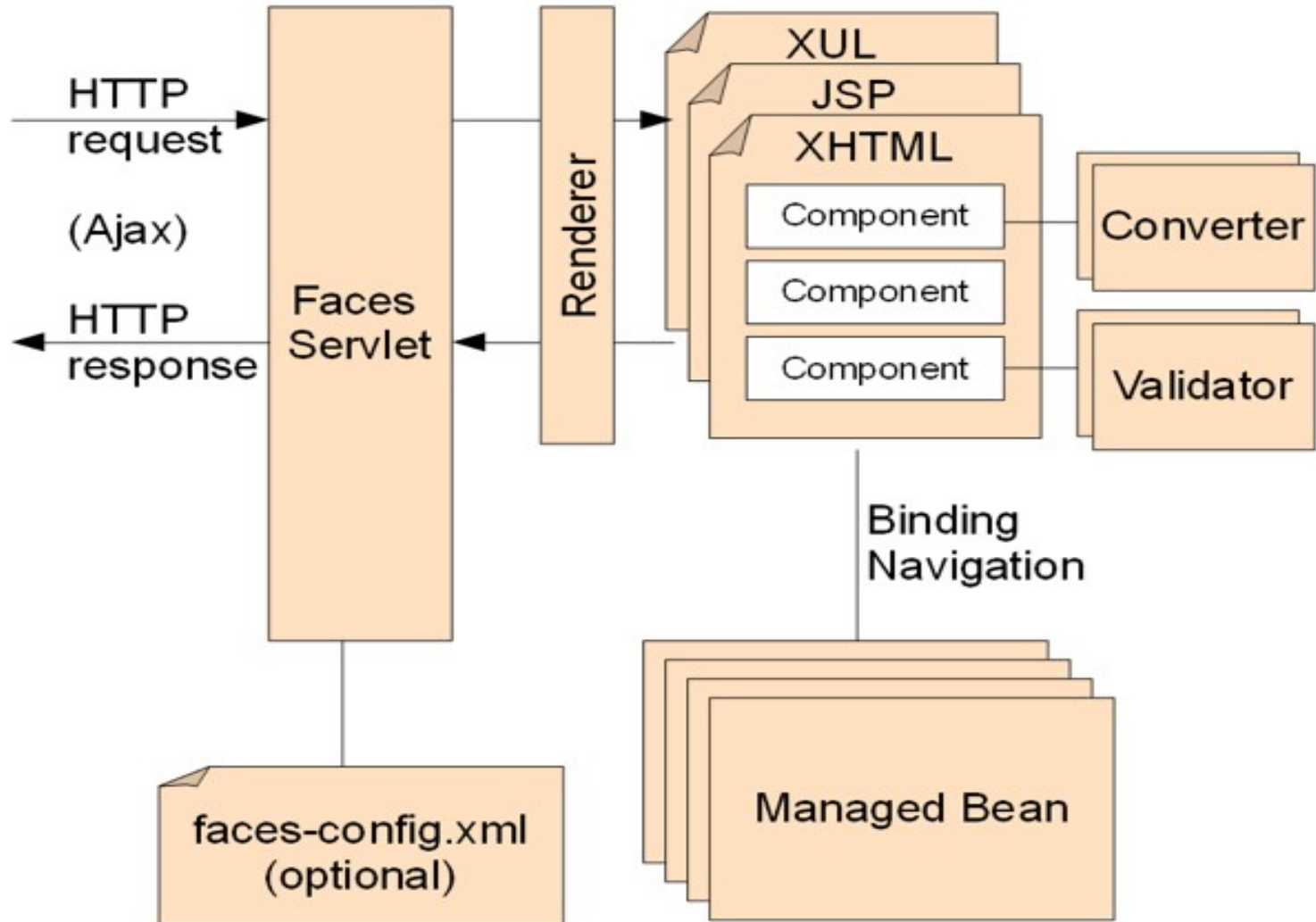
# Agenda

- Overview of EE 6 and GlassFish v3
- **Dive into some specs & demos**
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - **JSF 2.0**
  - Bean Validation 1.0
  - JAX-RS 1.1
  - CDI 1.0
- Summary

# JavaServer Faces (JSF) 2.0

- A component-oriented MVC framework
- Part of Java EE 6 and Web Profile
  - Other frameworks can rely on EE 6 extensibility
- Deserves its 2.0 version number
  - New features, issues fixed, performance focus
- Fully available today in Mojarra 2.0.x
  - Production-quality reference implementation
  - Part of GlassFish v3

# General JSF Architecture



# Binding

From xhtml to ManagedBean, and back

```
<h:inputText id="userName"  
    valueRef="#{loginForm.userName}" />  
<h:commandButton  
    action="#{loginForm.login}"  
    value="Login" />
```

```
@ManagedBean  
@RequestScope
```

```
public class LoginForm {  
    @ManagedProperty(value="anonymous")  
    private String userName;  
    public void setUsername(...) { ... }  
    public String getUsername() { ... }  
    public String login() {  
        return "success";  
    }  
}
```

anonymous Login

# Facelets now preferred VDL

- Facelets (XHTML) as alternative to JSP
  - Based on a generic View Description Language (VDL)
  - Can't add Java code to XHTML page (and “that's a good thing!”™)
- Pages are usable from basic editors
- IDEs offer traditional value-add:
  - Auto-completion (EL)
  - (Composite) Component management
  - Project management, testing, etc...

# Setup, configuration

- JSF 2.0 does not mandate Servlet 3.0
  - Servlet 2.5 containers will run JSF 2.0
  - `web.xml` may be optional depending on runtime
- `faces-config.xml` now optional
  - `@javax.faces.bean.ManagedBean`
  - *Not required with JSR 299*
  - Navigation can now belong to the page  
(`<navigation-rules>` become optional)

# Navigation

- JSF 1.x Navigation
  - Requires `faces-config.xml` editing

```
<navigation-rule>  
  <from-view-id>page1.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>next</from-outcome>  
    <to-view-id>/page2.xhtml</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

# Navigation

- JSF 1.x Navigation
  - Requires `faces-config.xml` editing
- Implicit Navigation (JSF 2.0)

```
<h:commandButton action="page2" value="Submit" />
```

```
<h:commandButton action="page2.xhtml" value="Submit" />
```

```
public String next() {  
    return "page2";  
}  
public String next() {  
    return "page2.xhtml";  
}
```

# Navigation

- JSF 1.x Navigation
  - Requires `faces-config.xml` editing

- Implicit Navigation (JSF 2.0)

```
<h:commandButton action="page2" value="Submit" />
<h:commandButton action="page2.xhtml" value="Submit" />
```

```
public String next() {
    return "page2";
}
public String next() {
    return "page2.xhtml";
}
```

- Conditional Navigation (JSF 2.0)

- New `<if>` tag in EL
- Declarative alternative to Java implementation in managed bean

```
<navigation-case>
    <from-outcome>next</from-outcome>
    <to-view-id>/page2.xhtml</to-view-id>
    <if>#{loginForm.isOkay}</if>
</navigation-case>
```

# Navigation

- JSF 1.x Navigation
  - Requires `faces-config.xml` editing

- Implicit Navigation (JSF 2.0)

```
<h:commandButton action="page2" value="Submit" />  
<h:commandButton action="page2.xhtml" value="Submit" />
```

```
public String next() {  
    return "page2";  
}  
public String next() {  
    return "page2.xhtml";  
}
```

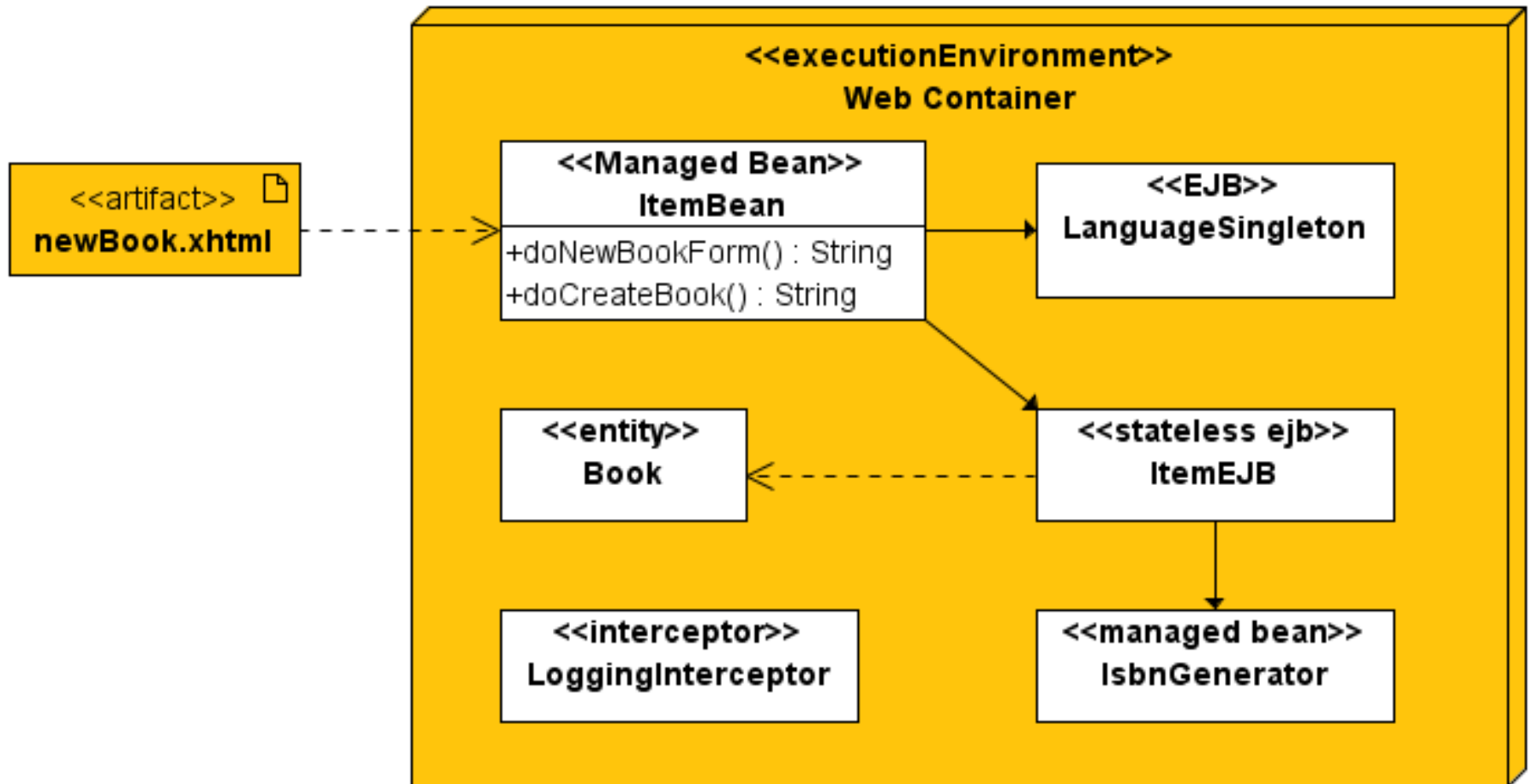
- Conditional Navigation (JSF 2.0)

- New `<if>` tag in EL
- Declarative alternative to Java implementation in managed bean

- Redirect (JSF 2.0)

```
<h:commandButton action="page2.xhtml?faces-redirect=true" value="Submit" />
```

# Demo : add a JSF page



# DEMO 07

Add a JSF page

# Validation

- Validators have been there since JSF 1.0

```
<h:inputText ... validator="#{bean.validateRuleXYZ}"/>
```

```
public void validateRuleXYZ(FacesContext context,  
    UIComponent toValidate, Object value) {  
    int input = (Integer) value;  
    if (input<min || input>max) {  
        ((UIInput) toValidate).setValid(false);  
        FacesMessage msg = new FacesMessage("Error");  
        context.addMessage(... , msg);  
    }  
}
```

# Validation (cont.)

- ... but the UI isn't the best place for this
- JSF integration with JSR 303 (BeanValidation)
  - Defines tier-independent constraints  
`@NotEmpty private String name;`
  - If JSR 303 implementation present, use it!
  - Default `javax.faces.Bean` validator added with `Application.addDefaultValidatorId()`
    - Causes every field to be validated on `UIInput.encodeEnd()`
    - Validation errors translated to JSF errors
    - Can be disabled on a component basis

# Complete JSF controller

```
@ManagedBean
@RequestScoped
public class LoginForm {

    @Pattern(regex="(.+)",
            message="{constraint.invalid.user}")
    private String userName;
    public void setUsername(String name) {
        this.userName = name;    }
    public String getUsername() {
        return userName;    }

    public String login() {
        return "page2.xhtml";    }

    public boolean isOkay() {
        if (whyNot) return false;
        else return true;
    }
}
```

# JSF Components

- Rather healthy component market
- Pretty good IDE support but...

# JSF Components

- Rather healthy component market
- Pretty good IDE support but...
- Building your own components with JSF 1.x was (much) harder than it should be
- *Bummer for an MVC “component” framework...*

# JSF Composite Component

- Using JSF 1.x
  - Implement `UIComponent`, markup in renderer, register in `faces-config.xml`, add `tld`, ...
- With JSF 2.0
  - Single file, no Java code needed
  - Use XHTML and JSF tags to create components

```
<html xmlns:cc="http://java.sun.com/jsf/composite">
```

```
<cc:interface>
```

```
  <cc:attribute ...>
```

```
<cc:implementation>
```

- Everything else is auto-wired

## `./web/resources/ezcomp/mycomponent.xhtml`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:composite="http://java.sun.com/jsf/composite">
  <!-- INTERFACE -->
  <composite:interface>
    <composite:attribute name="param"/>
  </composite:interface>
  <!-- IMPLEMENTATION -->
  <composite:implementation>
    <h:outputText value="Hello there, #{cc.attrs.param}"/>
  </composite:implementation>
</html>
```

## Using the component

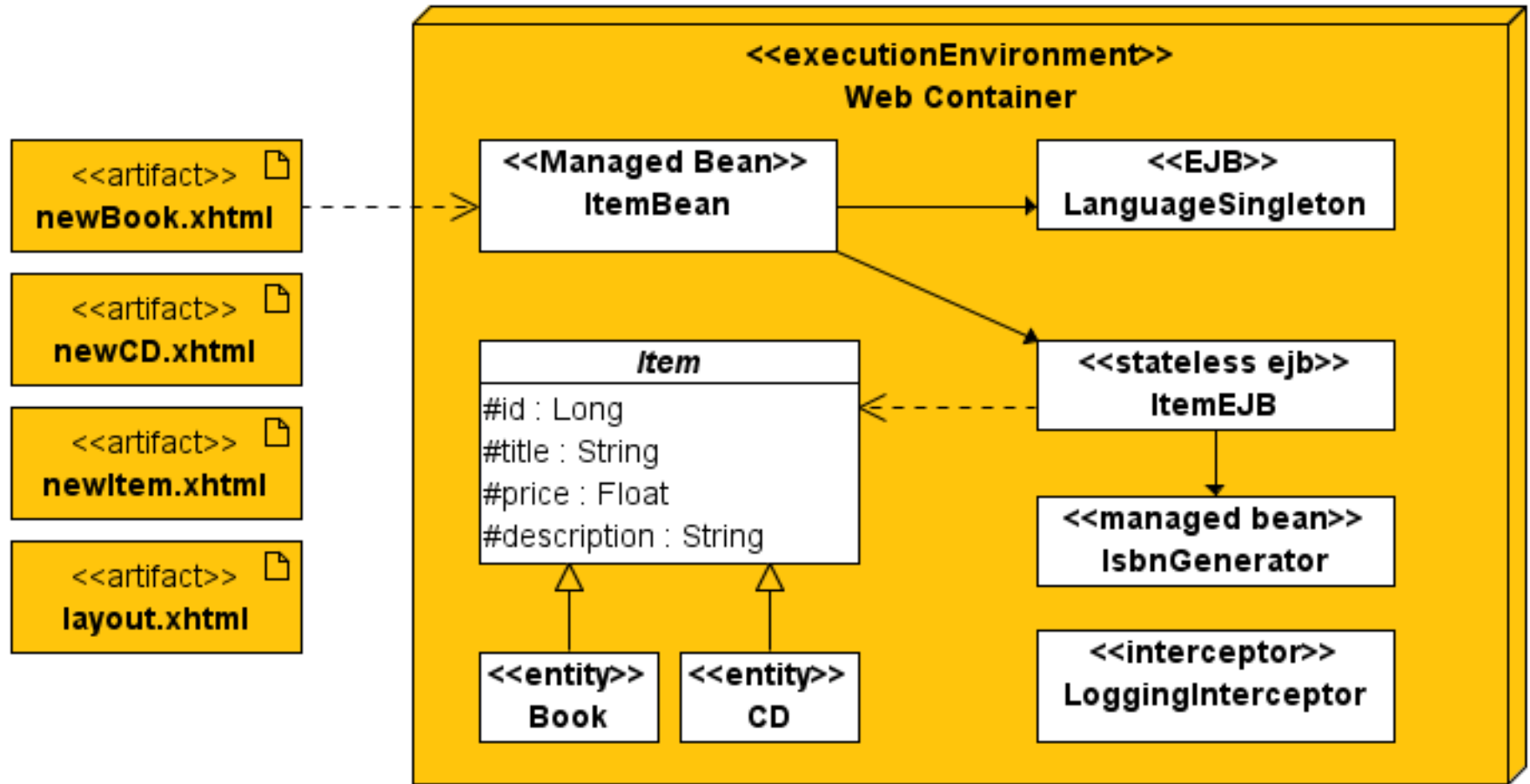
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:custom="http://java.sun.com/jsf/composite/ezcomp">
  <h:body>
    <custom:mycomponent param="Devoxx attendees"/>
  </h:body>
</html>
```

Defining the component

*Implicit EL object*



# Demo : 2 entities 1 component



# DEMO 08

Add a new CD entity with inheritance (Item)  
Add a new page with Composite Component

# Better Error Reporting

# Before...

```
javax.faces.FacesException: Expression Error: Named Object: euroConverter not found.
  at com.sun.faces.application.ApplicationImpl.createConverter(ApplicationImpl.java:1238)
  at com.sun.faces.facelets.tag.jsf.ConverterTagHandlerDelegateImpl.createConverter(ConverterTagHandlerDelegateImpl.java:148)
  at com.sun.faces.facelets.tag.jsf.ConverterTagHandlerDelegateImpl.applyAttachedObject(ConverterTagHandlerDelegateImpl.java:118)
  at javax.faces.view.facelets.FaceletsAttachedObjectHandler.applyAttachedObject(FaceletsAttachedObjectHandler.java:91)
  at com.sun.faces.facelets.tag.jsf.ConverterTagHandlerDelegateImpl.apply(ConverterTagHandlerDelegateImpl.java:73)
  at javax.faces.view.facelets.DelegatingMetaTagHandler.apply(DelegatingMetaTagHandler.java:114)
  at javax.faces.view.facelets.DelegatingMetaTagHandler.applyNextHandler(DelegatingMetaTagHandler.java:120)
  at com.sun.faces.facelets.tag.jsf.ComponentTagHandlerDelegateImpl.apply(ComponentTagHandlerDelegateImpl.java:204)
  at javax.faces.view.facelets.DelegatingMetaTagHandler.apply(DelegatingMetaTagHandler.java:114)
  at javax.faces.view.facelets.CompositeFaceletHandler.apply(CompositeFaceletHandler.java:91)
  at javax.faces.view.facelets.DelegatingMetaTagHandler.applyNextHandler(DelegatingMetaTagHandler.java:120)
  at com.sun.faces.facelets.tag.jsf.ComponentTagHandlerDelegateImpl.apply(ComponentTagHandlerDelegateImpl.java:204)
  at javax.faces.view.facelets.DelegatingMetaTagHandler.apply(DelegatingMetaTagHandler.java:114)
  at javax.faces.view.facelets.CompositeFaceletHandler.apply(CompositeFaceletHandler.java:91)
  at javax.faces.view.facelets.DelegatingMetaTagHandler.applyNextHandler(DelegatingMetaTagHandler.java:120)
  at com.sun.faces.facelets.tag.jsf.ComponentTagHandlerDelegateImpl.apply(ComponentTagHandlerDelegateImpl.java:204)
  at javax.faces.view.facelets.DelegatingMetaTagHandler.apply(DelegatingMetaTagHandler.java:114)
  at javax.faces.view.facelets.CompositeFaceletHandler.apply(CompositeFaceletHandler.java:91)
  at com.sun.faces.facelets.compiler.NamespaceHandler.apply(NamespaceHandler.java:86)
  at javax.faces.view.facelets.CompositeFaceletHandler.apply(CompositeFaceletHandler.java:91)
  at com.sun.faces.facelets.compiler.EncodingHandler.apply(EncodingHandler.java:75)
  at com.sun.faces.facelets.impl.DefaultFacelet.apply(DefaultFacelet.java:145)
  at com.sun.faces.application.view.FaceletViewHandlingStrategy.buildView(FaceletViewHandlingStrategy.java:715)
  at com.sun.faces.lifecycle.RenderResponsePhase.execute(RenderResponsePhase.java:106)
  at com.sun.faces.lifecycle.Phase.doPhase(Phase.java:101)
  at com.sun.faces.lifecycle.LifecycleImpl.render(LifecycleImpl.java:139)
  at javax.faces.webapp.FacesServlet.service(FacesServlet.java:311)
  at org.apache.catalina.core.StandardWrapper.service(StandardWrapper.java:1522)
  at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:279)
  at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:188)
  at org.apache.catalina.core.StandardPipeline.invoke(StandardPipeline.java:641)
  at com.sun.enterprise.web.WebPipeline.invoke(WebPipeline.java:97)
  at com.sun.enterprise.web.PESessionLockingStandardPipeline.invoke(PESessionLockingStandardPipeline.java:85)
  at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:185)
  at org.apache.catalina.core.StandardPipeline.invoke(StandardPipeline.java:641)
  at org.apache.catalina.connector.CoyoteAdapter.doService(CoyoteAdapter.java:329)
  at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:233)
  at com.sun.enterprise.v3.services.impl.ContainerMapper.service(ContainerMapper.java:161)
  at com.sun.grizzly.http.ProcessorTask.invokeAdapter(ProcessorTask.java:789)
  at com.sun.grizzly.http.ProcessorTask.doProcess(ProcessorTask.java:697)
  at com.sun.grizzly.http.ProcessorTask.process(ProcessorTask.java:951)
  at com.sun.grizzly.http.DefaultProtocolFilter.execute(DefaultProtocolFilter.java:166)
```

# ... after

## An Error Occurred:

Expression Error: Named Object: euroConverter not found.

+ Stack Trace

- Component Tree

```
<UIViewRoot id="j_id1" inView="true" locale="en_US" renderKitId="HTML_BASIC"
rendered="true" transient="false" viewId="/listBooks.xhtml">
```

```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
  <html xmlns="http://www.w3.org/1999/xhtml">
```

```
    <UIOutput id="j_idt3" inView="true" rendered="true" transient="false">
```

```
      <title>List of the books</title>
```

```
    </UIOutput>
```

```
  </UIViewRoot>
```

+ Scoped Variables

Oct 20, 2009 4:18:48 PM - Generated by Mojarra/Facelets

# Ajax support

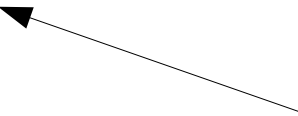
- Inspired by RichFaces, IceFaces, DynaFaces, ...
- Common JavaScript library (`jsf.js`)
  - request JavaScript functions captured by `PartialViewContext` for sub-tree processing
  - Client JavaScript updates the DOM

```
<h:commandButton  
    onclick="jsf.ajax.request(this,event,{render:'foo'});  
    return false;"/>
```

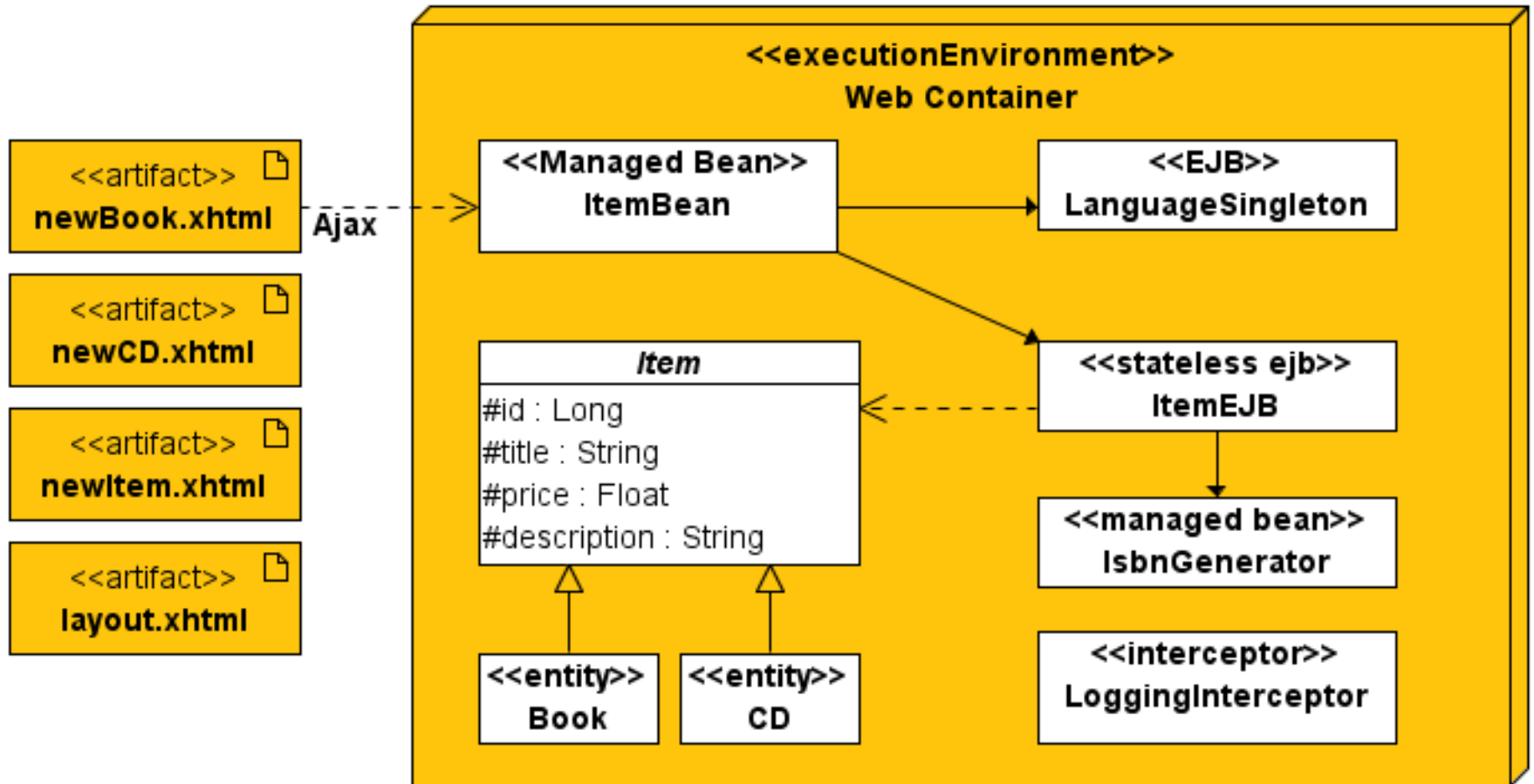
- `<f:ajax>` tag to ajaxify existing pages

```
xmlns:f="http://java.sun.com/jsf/core"  
<h:commandButton>  
    <f:ajax render="foo" execute="myForm" event="change" />  
</h:commandButton>
```

component `id(s)` to update or  
`@this, @none, @form, @all`



# Demo



# DEMO 09

Add Ajax calls

# NetBeans 6.8

# JSF Tooling

```
<br/>  
<h:outputText value="Hello there, #{cc.attrs.param} !"/>  
<h:form  
  <table border="0">  
    <tr>  
      <td>  
        <h:outputLabel value="ISBN : "/>  
      </td>  
      <td>  
        <h:inputText value="#{bookController.book.isbn}"/>  
      </td>  
    </tr>  
  </table>  
</h:form>
```

isbn String

```
<h:column>  
  <f:facet name="header">  
    <h:outputText value="Description"/>  
  </f:facet>  
  <h:outputText value="#{bk.description}"/>  
</h:column>  
  
<h:column>  
  <f:facet name="header">  
    <h:outputText value="Number of Pages"/>  
  </f:facet>  
  <h:outputText value="#{bk.nbOfPage}"/>  
</h:column>  
  
<h:column>  
  <f:facet name="header">  
    <h:outputText value="Number of Reviews"/>  
  </f:facet>  
  <h:outputText value="#{bk.nbOfReviews}"/>  
</h:column>  
</h:dataTable>  
<h:form>
```

- View
- Format
- Cut
- Copy
- Paste
- Code Folds
- Convert to Composite Component
- Select in

Project: login.xhtml x

Structure: composite:interface

```
<composite:interface>  
  <composite:actionSource name="loginButton" targets="form:loginButton" />  
  
  <composite:attribute name="loginButtonText" default="Log In" required="true" />  
  <composite:attribute name="loginPrompt" />  
  <composite:attribute name="namePrompt" />  
  <composite:attribute name="passwordPrompt" />  
  <composite:attribute name="loginAction" method="signature"="java.lang.String action()" />  
  <composite:attribute name="managedBean" />  
</composite:interface>
```

Structure: masterLayout.xhtml x

```
<div class="pageHeading">  
  <ui:insert name="heading">  
    #{msgs.placesHeading}  
  </ui:insert>  
</div>  
<div class="menuAndContent">  
  <div class="menuLeft">  
    <ui:insert name="menuLeft" />  
  </div>  
  <div class="content" style="display: inline-block; vertical-align: top; width: 100%;">  
    <ui:insert name="content" />  
  </div>  
</div>  
<div class="menuRight">  
  <ui:insert name="menuRight" />  
</div>
```

Structure: places.xhtml x

```
<ui:composition xmlns="http://www.w3.org/2001/XMLSchema-instance" xmlns:ui="http://java.sun.com/jsf/facelets" template="/templates/masterLayout.xhtml">  
  <ui:define name="menuLeft">  
    <ui:include src="/sections/places/menuLeft" />  
  </ui:define>  
  <ui:define name="content">  
    <ui:include src="/sections/places/content" />  
  </ui:define>  
  <ui:define name="menuRight">  
    <ui:include src="/sections/places/menuRight" />  
  </ui:define>  
</ui:composition>
```

# IntelliJ Maia (v9)

Did you know that Quick Documentation View (Ctrl+Q) works in completion look

# And more...

- Validation delegated to BeanValidation
- Easier resources management
- New managed bean scope (View)
- Groovy support (Mojarra)
- Bookmarkable URLs
- Templating : define and apply layouts
- Project stages (dev vs. test vs. production)
- ...

# Agenda

- Overview of EE 6 and GlassFish v3
- Dive into some specs & demos
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - Bean Validation 1.0
  - JAX-RS 1.1
  - CDI 1.0
- Summary



# Bean Validation 1.0

- Enable declarative validation in your applications
- Constrain Once, Validate Anywhere
  - restriction on a bean, field or property
  - not null, size between 1 and 7, valid email...
- Standard way to validate constraints
- Integration with JPA 2.0 & JSF 2.0

# Bean Validation 1.0

```
public class Address {  
    @NotNull @Size(max=30,  
        message="longer than {max} characters")  
    private String street1;  
    ...  
    @NotNull @Valid  
    private Country country;  
}
```

```
public class Country {  
    @NotNull @Size(max=20)  
    private String name;  
    ...  
}
```

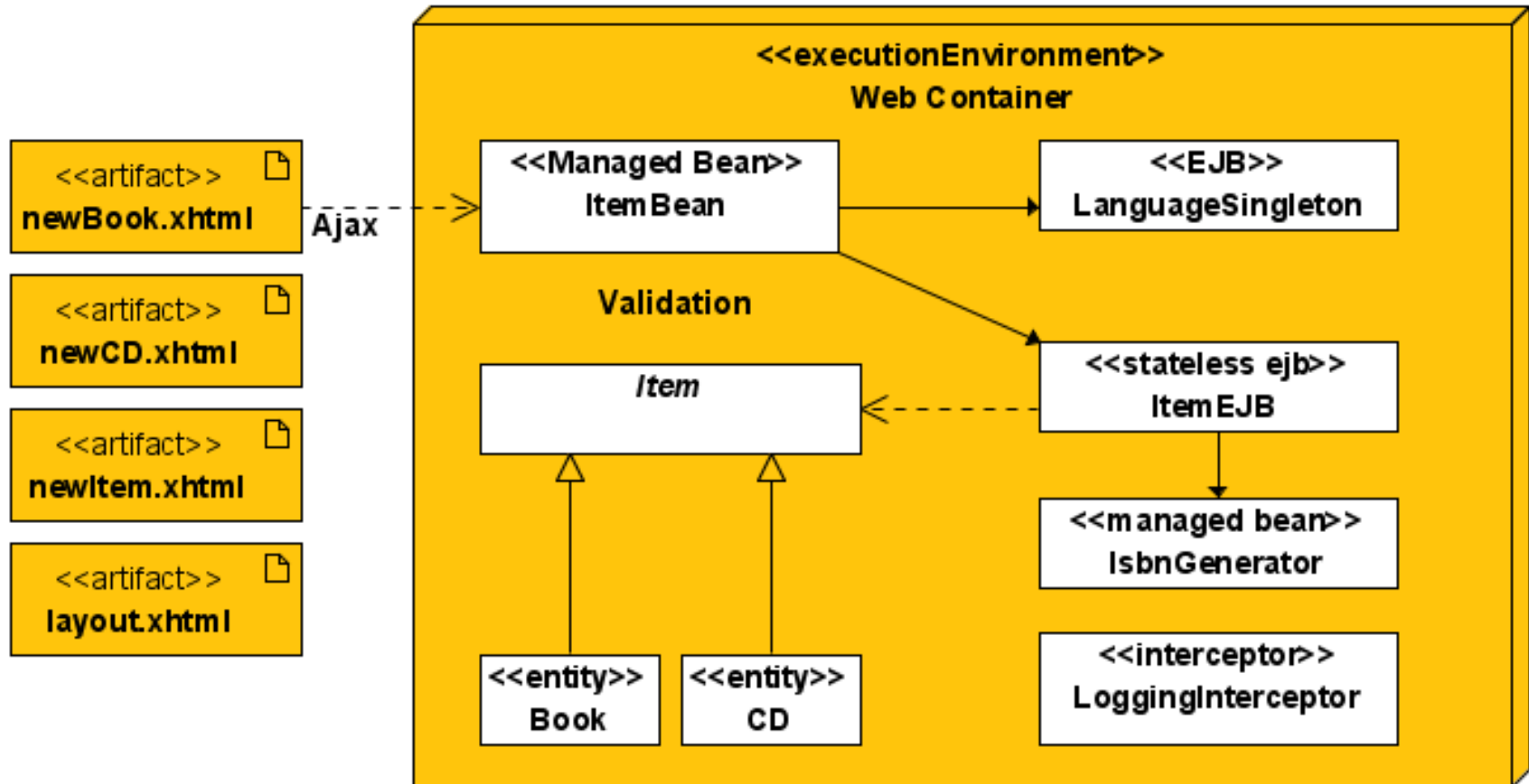
request recursive  
object graph  
validation



# Build your own!

```
@Size(min=5, max=5)
@ConstraintValidator(ZipcodeValidator.class)
@Documented
@Target({ANNOTATION_TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface ZipCode {
    String message() default "Wrong zipcode";
    String[] groups() default {};
}
```

# Demo : Validation Item/ItemBean



# DEMO 10

Add some validation on  
Item entity and ItemBean

# And more...

- Group subsets of constraints
- Partial validation
- Order constraint validations
- Create your own `@Constraint`
- Bootstrap API
- Messages can be i18n
- ...

# Agenda

- Overview of EE 6 and GlassFish v3
- **Dive into some specs & demos**
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - Bean Validation 1.0
  - **JAX-RS 1.1**
  - CDI 1.0
- Summary

# JAX-RS 1.1

- High-level HTTP API for RESTful Services
- POJO and Annotations Based
  - API also available
- Maps HTTP verbs (Get, Post, Put, Delete...)
- JAX-RS 1.0 has been released in 2008
- JAX-RS 1.1 integrates with EJBs  
(and more generally with Java EE 6)

# Hello World

```
@Path("/helloworld")  
public class HelloWorldResource {  
  
    @GET  
    @Produces("text/plain")  
    public String sayHello() {  
        return "Hello World";  
    }  
}
```

*<http://example.com/helloworld>*



# Hello World

## Request

---

```
GET /helloworld HTTP/1.1  
Host: example.com  
Accept: text/plain
```

## Response

---

```
HTTP/1.1 200 OK  
Date: Wed, 12 Nov 2008 16:41:58 GMT  
Server: Apache/1.3.6  
Content-Type: text/plain; charset=UTF-8  
Hello World
```

# Different Mime Types

```
@Path("/helloworld")
public class HelloWorldResource {

    @GET @Produces("image/jpeg")
    public byte[] paintHello() {
        ...
    }
    @GET @Produces("text/plain")
    public String displayHello() {
        ...
    }
    @POST @Consumes("text/xml")
    public void updateHello(String xml) {
        ...
    }
}
```

# Parameters & EJBs

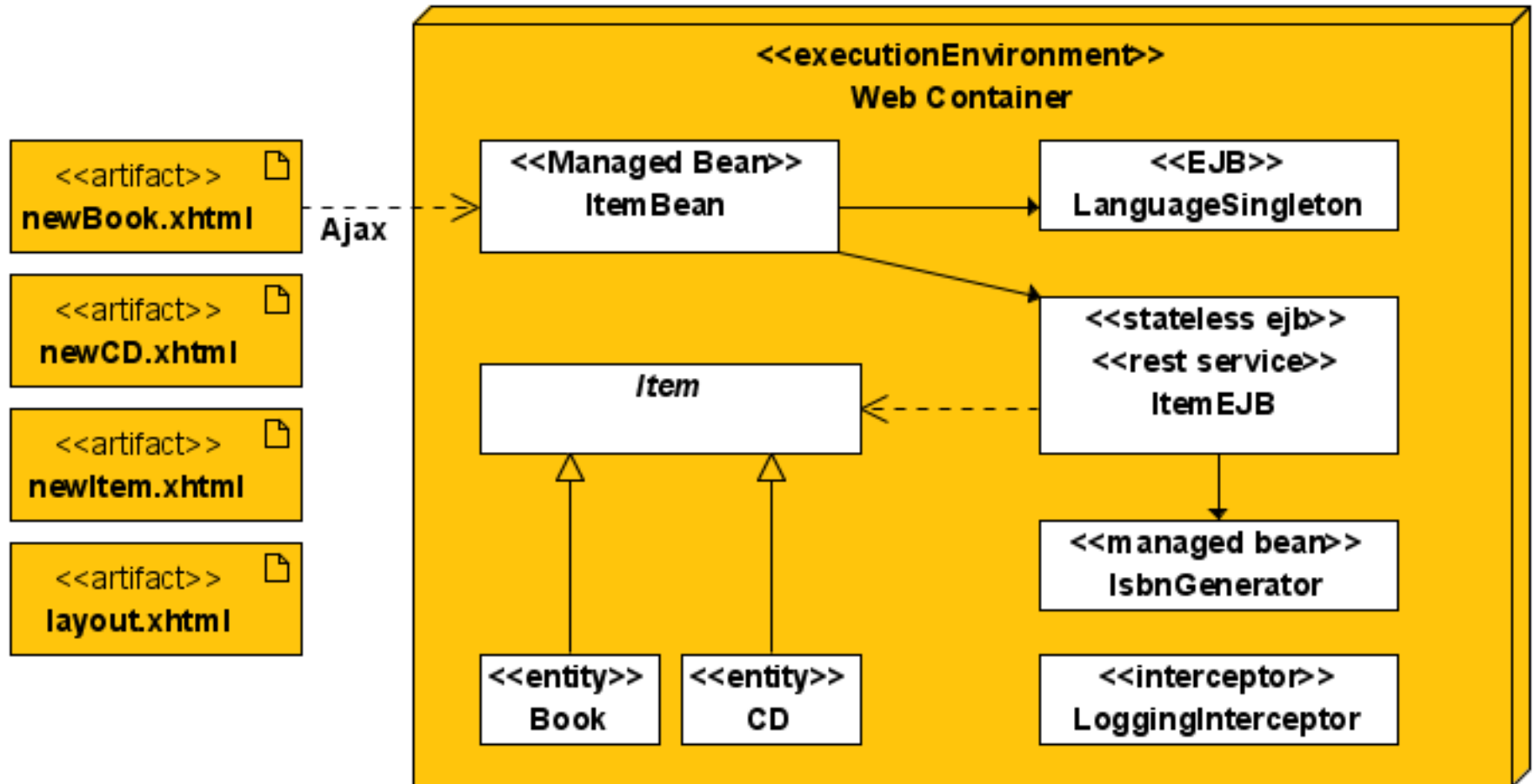
```
@Path("/users/{userId}")
@Stateless
public class UserResource {

    @PersistenceContext
    EntityManager em;

    @GET @Produces("text/xml")
    public String getUser(@PathParam("userId")
                          String id) {

        User u = em.find(User.class, id)
        ...
    }
}
```

# Demo : Add REST service to EJB



# DEMO 11

Add a REST service to the ItemEJB

# And more...

- **Different parameters** (`@MatrixParam`,  
`@QueryParam`, `@CookieParam` ...)
- **Support for** `@Head` **and** `@Option`
- **Inject** `UriInfo` **using** `@Context`
- **No** `web.xml` **using**
  - `@ApplicationPath("rs")`  
on `javax.ws.rs.core.Application`
  - **Can be overridden with** `web.xml`
- **Providers**
- ...

# Agenda

- Overview of EE 6 and GlassFish v3
- Dive into some specs & demos
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - Bean Validation 1.0
  - JAX-RS 1.1
  - CDI 1.0
- **Summary**

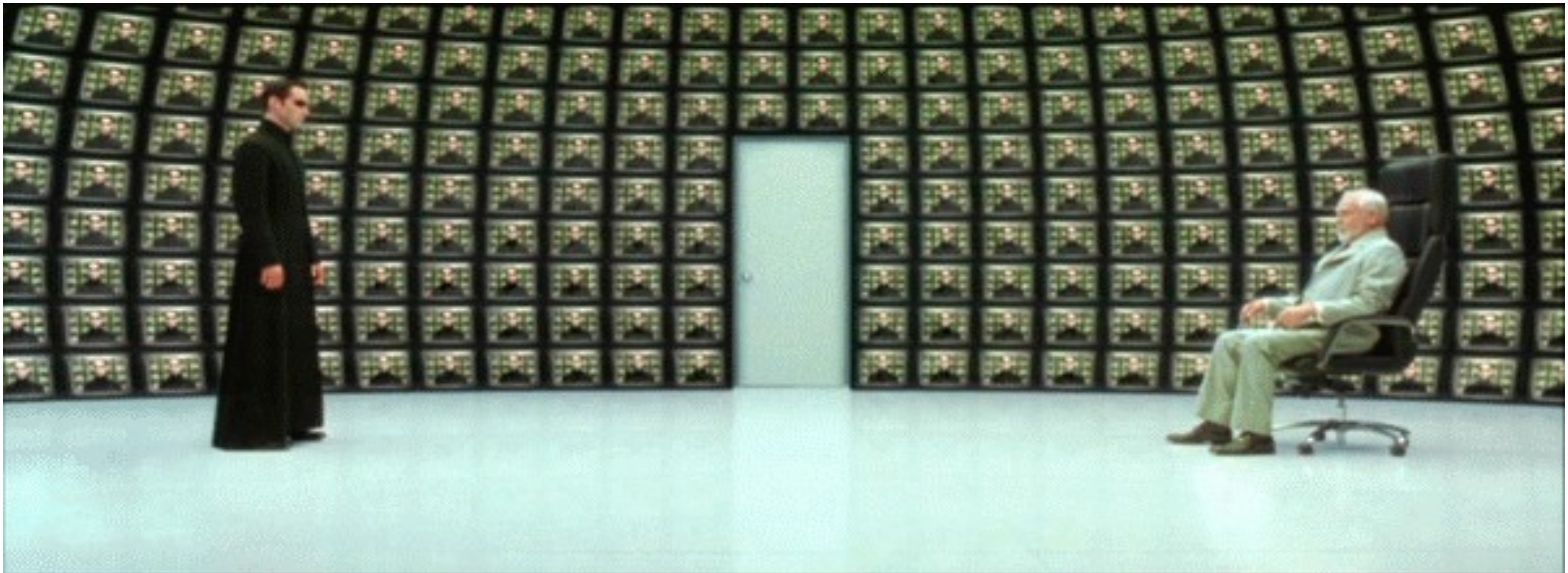


# Demo : give me some numbers !

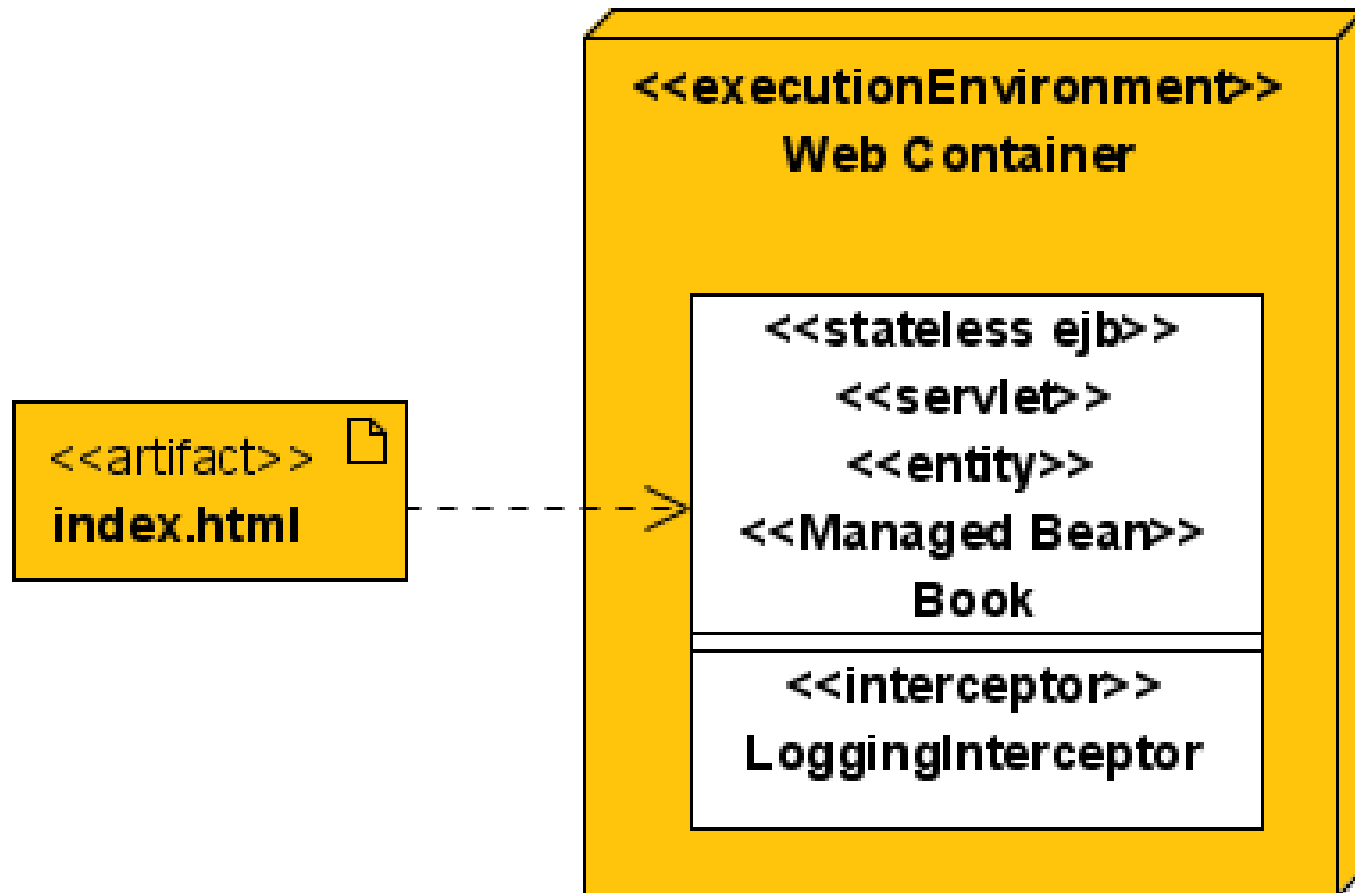
- The final demo application is :
  - 8 classes, 0 interface
  - 366 LOC of Java  
(with getters/setters, multiple imports, ...)
  - 12 LOC of XML in `persistence.xml`
  - 274 LOC of XHTML  
(2 pages, 1 component, 1 layout)

# You can always do what you want

- Java EE 6 is simple
- Java EE 6 is rich
- Mix all the technologies in one layer
- Ask the architect



# Demo : a Monster EJB



# DEMO 12

An EJB that does it all

# Agenda

- Overview of EE 6 and GlassFish v3
- Dive into some specs & demos
  - JPA 2.0
  - Servlet 3.0
  - EJB 3.1
  - JSF 2.0
  - Bean Validation 1.0
  - JAX-RS 1.1
  - CDI 1.0
- **Summary**



# Injection in Java EE 5

- **Common Annotation**
  - `@Resource`
- **Specialized cases**
  - `@EJB`, `@WebServicesRef`,  
`@PersistenceUnit` ...
- **Requires *managed* objects**
  - EJB, Servlet and JSF Managed Bean in EE 5
  - Also in any Java EE 6's  
`javax.annotation.ManagedBean`

# Injection in Java EE 6

CDI (JSR 299)  
&  
DI (JSR 330)

*Inject just about anything anywhere...  
...yet with strong typing*

# The tale of 2 dependency JSRs

- Context & Dependency Injection for Java EE
  - Born as WebBeans, unification of JSF and EJB
  - “Loose coupling, strong typing”
  - JBoss Seam as strong influencer and spec lead
  - Weld as the reference implementation
- Dependency Injection for Java (JSR 330)
  - Lead by Google and SpringSource
  - Minimalistic dependency injection, `@Inject`
  - Applies to Java SE, Guice as the reference impl.
- Both aligned and part of Java EE 6 Web Profile

# CDI in a slide

## Component Management Services

- Better lifecycle for stateful objects, bound to well-defined contexts
  - *Managed Beans & (numerous) additional services*
- Typesafe dependency injection (no XML)
- Interactions via events
- Interceptors ++
  - New kind of interceptor (decorator)
  - New binding approach
- An SPI for portable extensions to the container

# @Named **and** @Inject

- **CDI requires a WEB-INF/beans.xml file**
  - Can be empty
  - Beans auto-discovered at startup
- **@Named makes the bean available to EL**
  - **Prefer @Named to @ManagedBean (JSF or JSR 250)**
- **Use @Inject to inject :**
  - `@Inject IsbnGenerator generator;`
- **@Resource still around**
  - Use it for DB connexions, queues, RA's
  - Anything App-managed: use @Inject

# DEMO 13

Enable CDI and replace `@Resource` with `@Inject`

**qualifier** (user-defined label)  
i.e. « which one? »

**@Inject** **@Premium** **Customer** cust;

**injection point**

**type**

# Qualifier Annotation

```
@Target ({TYPE, METHOD, PARAMETER, FIELD})
```

```
@Retention (RUNTIME)
```

```
@Documented
```

```
@Qualifier
```

```
public @interface Premium {...}
```

```
@Premium // my own qualifier (see above)
```

```
public class SpecialCustomer
```

```
    implements Customer {
```

```
    public void buy() {...}
```

```
}
```

# DEMO 14

Use CDI qualifiers (and events)

# Constructor and init Injections\*

**@Inject**

// one constructor only

```
public Order (@Premium Customer cust) {  
    ...  
}
```

**@Inject**

// random init method

```
void populate (@Premium Customer cust) {  
    ...  
}
```

*\*: new in Java EE*

# Contexts

The 'C' in CDI

- **Built-in “Web” Scopes :**

- `@RequestScoped`
- `@SessionScoped*`
- `@ApplicationScoped*`
- **`@ConversationScoped*`**

\*: requires `Serializable` fields to enable passivation

- **Other Scopes**

- `@Dependent` is the default pseudo-scope for un-scoped beans (*same as Managed Beans*)
- Build your own `@ScopeType`

- **Clients need not be scope-aware**

# @ConversationScoped

- *A conversation* is :
  - explicitly demarcated
  - associated with individual browser tabs
  - accessible from any JSF request

```
@Named
```

```
@ConversationScoped
```

```
public class ItemFacade implements Serializable {  
    @Inject Conversation conversation;  
    ...  
    conversation.begin(); // long-running  
    ...  
    conversation.end(); // schedule for destruction
```

# DEMO 15

Use CDI conversation scope

# Producer Methods

*and fields too !*

**@Produces**

**@MaxNumber**

```
int getMaxNumber() {  
    return 100;  
}
```

```
// somewhere else
```

```
@Inject
```

```
@MaxNumber // no class dependency
```

```
private int maxNumber;
```

# Various

- CDI from a Servlet :

```
public class Login extends HttpServlet {  
    @Inject Credentials credentials;  
    @Inject Login login;  
}
```

- Similar integration with other Java EE APIs
- Other CDI implementations:
  - CanDI @ Caucho
  - OpenWebBeans @ Apache

# But Wait! There's more...

- **Alternatives**
  - `@Alternative` annotation on various impl.
  - `beans.xml` to declare which one to use on deploy
- **Interceptors & Decorators**
  - Loosely-coupled orthogonal (technical) interceptors
  - `@Decorator` bound to given interface
- **Stereotypes (`@Stereotype`)**
  - Captures any of the above common patterns
- **Events**
  - Loosely-coupled (conditional) `@Observable` events
- **BeanManager API (Injection metamodel)**
  - Define/modify beans and injection points
  - The sky is the limit !

# To learn more about CDI

- *Not (yet) covered in Antonio's book*
- The CDI specification is terse (92 pages) but more aimed at implementers
- Try one of the following :
  - Java EE 6 tutorial (Part V)
  - JBoss Weld documentation
  - Java EE 6 SDK Samples
  - Java EE 6 & GlassFish v3 Virtual Conference

# Summary

- You've quickly seen
  - New concepts
  - New specifications
  - New features on existing specifications
- Want to know more ?

# Thanks for your attention!

- <http://java.sun.com/javaee>
- <http://jcp.org/en/jsr/summary?id=316>
- Java EE 6 and GlassFish v3 Virtual Conference  
<http://www.sun.com/events/javaee6glassfishv3/virtualconference/index.jsp>
- “Introducing the Java EE 6 Platform” article  
<http://java.sun.com/developer/technicalArticles/JavaEE/JavaEE6Overview.html>
- <http://glassfish.org>
- <http://beginningee6.kenai.com/>

# The JavaEE 6 Platform



[alexis.mp@sun.com](mailto:alexis.mp@sun.com)

<http://blog.sun.com/alexismp>

twitter:alexismp

[antonio.goncalves@gmail.com](mailto:antonio.goncalves@gmail.com)

<http://agoncal.wordpress.com>

twitter:agoncal

