A Component Based Approach

JSF Slides #5

Characterization Review

Much more of standard (non-web) OO-programming

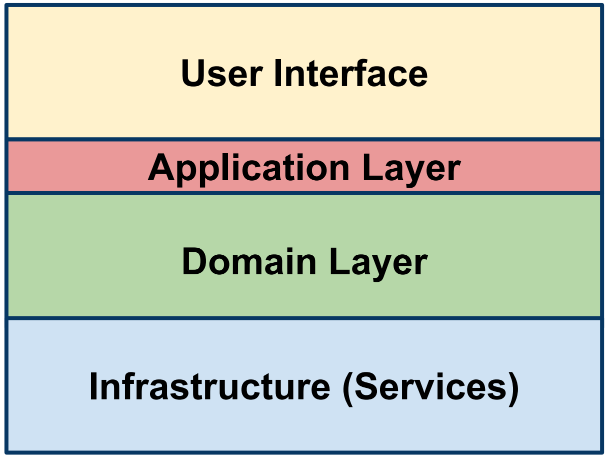
- Well known concepts of objects, components and listeners
- High abstraction level avoid accessing HTTP request etc. (but pops-up...)
- Libraries of GUI components (higher level frameworks)
- Possibly a bit lack of control

Design and MVC

JSF/Facelets/CDI/Bean Validation is not a complete framework, no default MVC design

- We use no framework
- For now we have to design ourselves, so yet another in house MVC- solution

Application Layers vs JEE Stack



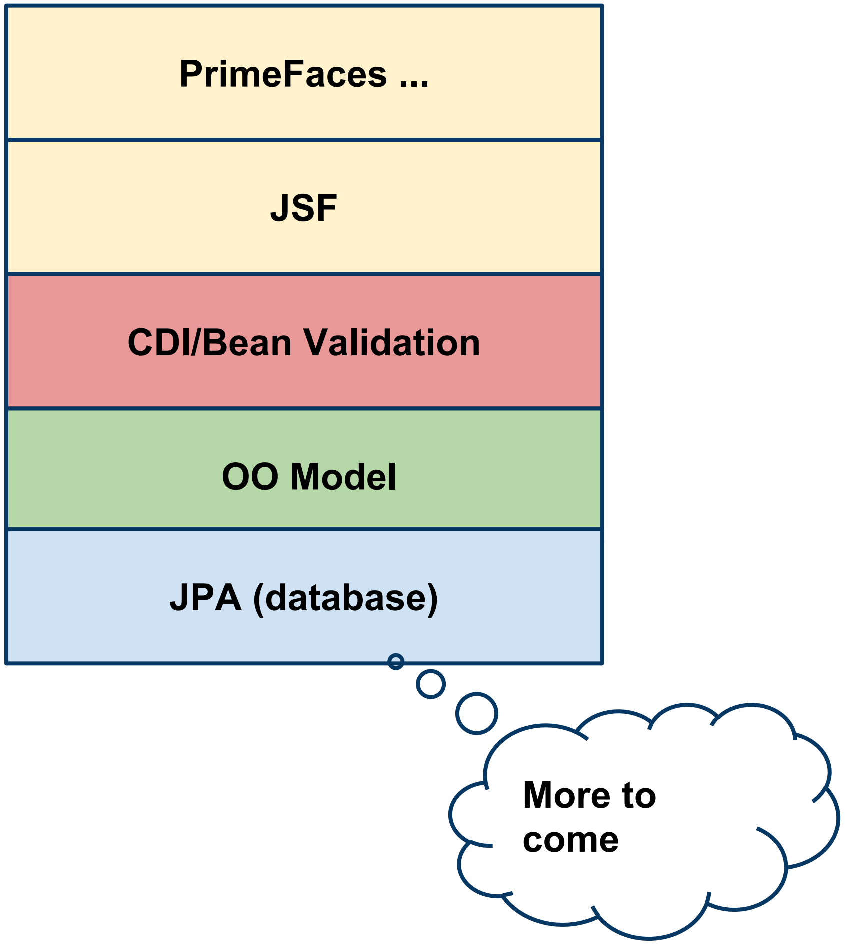
User Interface

Application Layer

Domain Layer

Infrastructure (Services)

Domain driven application layering



PrimeFaces ...

JSF

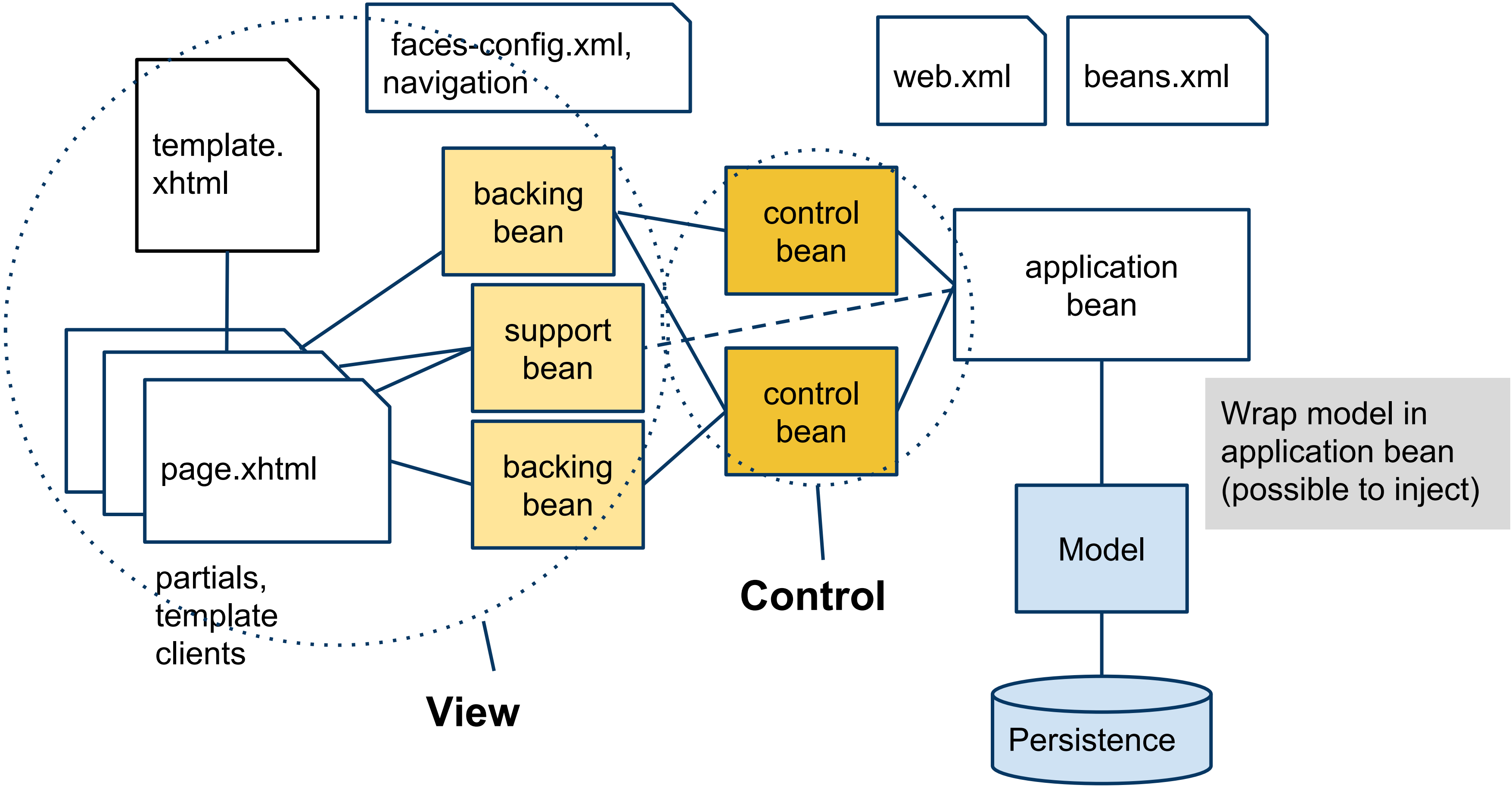
CDI/Bean Validation

OO Model

JPA (database)

More to come

JSF Application Design



Wrap model in application bean (possible to inject)

- - - - - - = read only

NOTE: We don’t have any persistence layer yet (database)

The “Master/Detail” Problem

Solved by (similar to request based approach)

```
<!--In a master table -->
<td>
   <h:link value="Edit" outcome="personDetail">
      <f:param name="id" value="#{person.id}" />
      <f:param name="fname" value="#{person.fname}" />
      <f:param name="age" value="#{person.age}" />
   </h:link>
</td>

<!-- Detail page -->
<f:metadata>
   <f:viewParam name="id" value="#{personDetail.id}"  />
   <f:viewParam name="fname" value="#{personDetail.fname}"  />
   <f:viewParam name="age" value="#{personDetail.age}"  />
</f:metadata>
```

Pagination

One possibility for list
- ViewScoped bean holds currentPage, methods prev, next, … call with AJAX (some method in request cycle, to get new values)

```
<!-- Current page in personList bean -->
<h:commandButton value="Prev" actionListener="${personList.prev}">
     <f:ajax execute="@form" render="personsPanel" />
</h:commandButton>

<h:commandButton value="Next" actionListener="${personList.next}" >
      <f:ajax execute="@form" render="personsPanel" />
</h:commandButton>
```

Or SessionScoped bean with non-AJAX calls or higher level component suites

JSF PRG

PRG pattern

- Use view parameters
- If no data should survive use redirect (see navigation)
- If data should survive use “includeViewParams=true”
- See code samples

JSF Resources

For CSS, images, JavaScript, ...using the "library" attribute

Example: NetBeans project/Maven
CSS in Web Pages/resources/css, JavaScrip in Web Pages/resources/js, etc

```
<html>
 <h:outputStylesheet library="css" name="styles.css" />
 <h:outputScript library="js" name="utils.js" target="head"/>
   <body>
     <h:graphicImage library="img"
                               name="tomato.jpeg" alt="tomato"/>
```

JSF I18N

Internationalization (i18n) using resource bundles
- msg.properties, msg_de.properties, msg_sv.properties, ...
- Flat text files to Map<String, String>

In faces-config.xml

```
<resource-bundle>
    <!-- Package and folder hierarchy (must be nested)-->
   <base-name>edu.chl.hajo.i18n.msg</base-name>
   <!-- This is the name used in EL-expressions -->
   <var>msg</var>
</resource-bundle>
```

In page

```
    <..."#{msg.lblWelcome}".../>
```

JSF and Cleans URI's

No standard... as noted before

Have to rely on third party, PrettyFaces, others..

Testing

Beans must run in container, how to test?

- If using constructor or method injection possible to supply needed objects. Run JUnit test like POJO's
- Better use embedded container ("Arquillian" , see database slides)

Some thought's

- If using CDI as a thin administrative layer between GUI and model, there should not be much of testing needed
- No application logic in pages, beans

Authentication and Authorization

Standard JEE authorization technique, using realms

A **realm** is a security policy domain defined for a web or application server. A realm contains a collection of users, who may or may not be assigned to a group

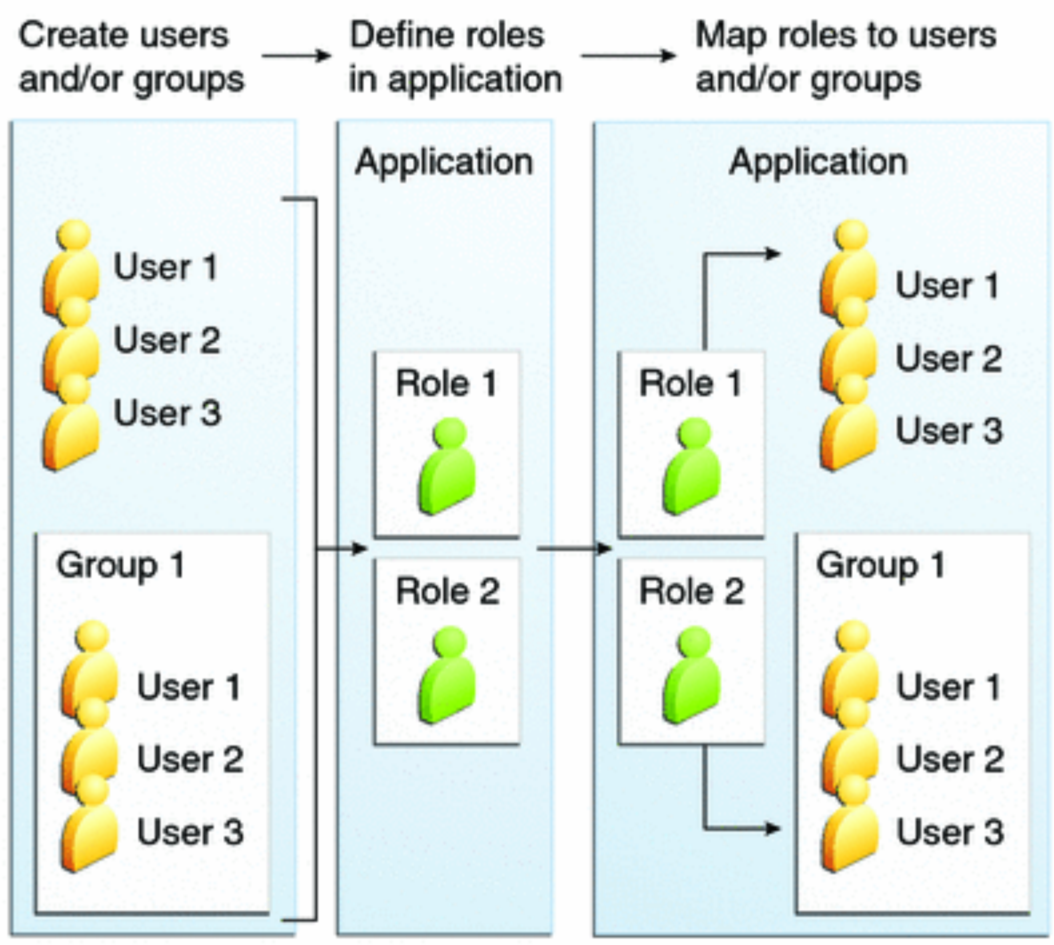
Types of realms (supported by GlassFish and Tomcat)
- **file**, Stores user information in a file. This is the default realm when you first install the GlassFish Server
- **ldap**, Stores user information in an LDAP directory
- **jdbc**, Stores user information in a database

…

Realms, Users, Groups, and Roles

A realm contains a collection of users, who may or may not be assigned to a group

A role is an abstract name for the permission to access a particular set of resources in an application.



File realm

For now we use the file realm with GlassFish
- This is server dependent (Tomcat different)
- Database backed sample later (better)

Steps
-Create users and groups in GlassFish file realm (using Admin console) …
-Create roles in application (defined in glassfish-web.xml)
-Map roles to users and groups (also glassfish-web.xml)
-Specify security constraints in web.xml

Web Security Constraints

Web resource collection: A list of URL patterns (the part of a URL after the hostname and port you want to constrain) and HTTP operations (the methods within the files that match the URL pattern you want to constrain) that describe a set of resources to be protected.

Authorization constraint: Specifies whether authentication is to be used and names the roles authorized to perform the constrained requests.

User data constraint: Specifies how data is protected when transported between a client and a server.

Web Security Constraints Example

// web.xml

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>wholesale</web-resource-name>
        <url-pattern>/acme/wholesale/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>PARTNER</role-name>  <!-- Role name in application -->
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

CONFIDENTIAL = GlassFish will use SSL (alt. NONE)

Web Authorization Mechanism

JEE supports

-Basic authentication (demo at service based approach)
-Form-based authentication (at component based approach)
-Digest authentication
-Client authentication
-Mutual authentication

Now we'll use Form-Based (preferred way to so it)

Programmatic Login

Use a <h:form> for login and password
- Navigation see code samples

```
// In some backing bean connected to login page
// Using default mechanism and HTTPServletRequest (request)
try {
   request.login(id, password);
   User user = userService.find(id, password);
   externalContext.getSessionMap().put("user", user);
   return "success";
} catch (ServletException e) {
   FacesContext.getCurrentInstance().addMessage(null,
                  new FacesMessage(FacesMessage.SEVERITY_WARN,
                                  "Login Failed", null));
   externalContext.getFlash().setKeepMessages(true);
}
return "fail";
```