Java EE Architecture, Part One

Content

- The Layer pattern
- Layers in a Java EE application
- The client layer
- Framework overview
- Do not write infrastructure code
- Requirements on the Presentation layer
- Security
- Frameworks for the Presentation layer

A Short Repetition of the Layer Pattern

- Problems:
  - Business logic should not be mixed with the view.
  - Services (like persistence) should not be mixed with business logic.
  - Need low coupling between subsystems.
  - Components and subsystems should be easy to modify or change.
  - The code should be easy to understand.
  - The code should be easy to modify and extend.

A Short Repetition of the Layer Pattern, Cont'd

- Solution:
  - Divide the system in layers. Layers are the coarsest parts of the system and a layer typically consist of several components, packages and subsystems.
  - Each layer should have high cohesion.
  - Only allow coupling from higher (closer to the user interface) layers to lower, since higher layers are less stable and lower layers do not need higher ones.
  - If possible, only allow coupling to the nearest lower layer.

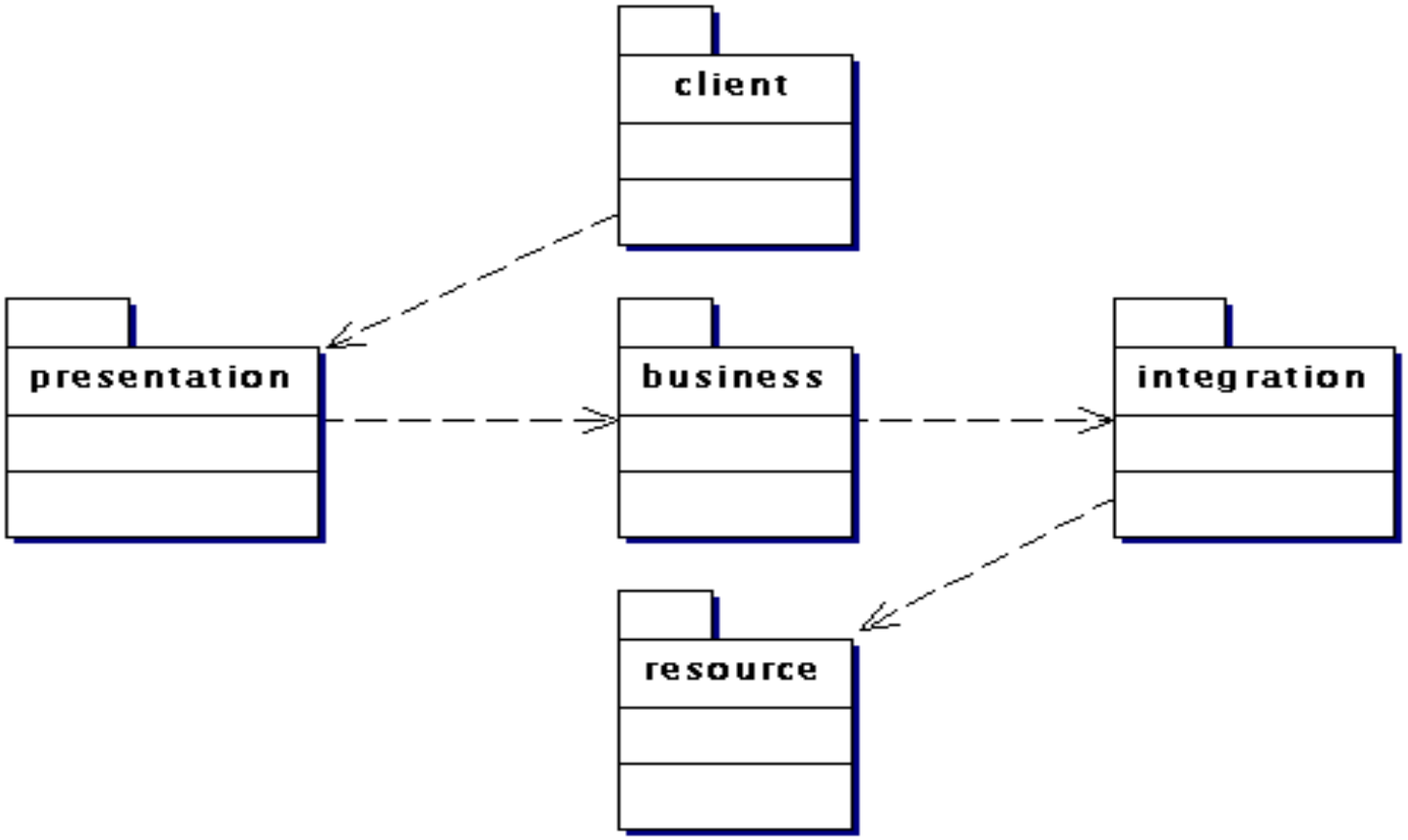
A Short Repetition of the Layer Pattern, Cont'd

- Advantages:
  - Higher cohesion and lower coupling.
  - Easier to reuse components both in the same application and between applications.
  - Easier to divide responsibility between developers.
  - Easier to distribute the system on different nodes.
  - Easier to change, extend and modify subsystems.

A Short Repetition of the Layer Pattern, Cont'd

- Disadvantages:
  - Might be difficult to decide which layers are needed. Too many layers means unnecessary work.
  - Might cause performance problems. The problem here is how many interprocess calls are made, ordinary method calls take very little time.

Layers in a Java EE Application



The Client Layer

- Browser
  - Easiest kind of client since browsers are without unknown bugs and since there will be little problem with firewalls.
  - Use browsers as clients if there are no very good reason for another choice.
  - The only possible client if the application is to be used by “anyone who happens to surf by”.

The Client Layer, Cont'd

- Browser, Cont'd
  - Necessary to decide which browsers to support (both product and version). This is a non-functional requirement that must be decided early and tested throughout the entire development.

The Client Layer, Cont'd

- Browser with applet or Java Web Start
  - Lots of things can be done with HTML, CSS, Java Script etc.
  - Applets and Java Web Start cause problems since most browsers are delivered without Java Support and downloading and installing a JRE requires quite a lot from the user.
  - None are a good idea if the application is meant for anyone.

The Client Layer, Cont'd

- Browser with applet or Java Web Start, Cont'd
  - If it is important to run a program on the user's computer is it most likely better to choose Java Web Start than applets.
  - Java Web Start allows the user to store the program locally instead of contacting the server every time it shall be used.
  - The conclusion is that applets are “out”.

The Client Layer, Cont'd

- Fat client
  - An ordinary Java (or other language) program that opens a network connection to the Java EE server.
  - Might be a choice for a very closed community, for example staff using a company's intranet.
  - Use only if really necessary.
  - Why not use Java Web Start to distribute and install the clients?

The Client Layer, Cont'd

- Java EE client (application client)
  - Part of the Java EE specification.
  - Can be packaged in an ear together with the Java EE application.
  - Has deployment descriptor like web components and EJBs.
  - Has access to some Java EE apis like JMS and web services.

The Client Layer, Cont'd

- Java EE client (application client) , Cont'd
  - The specification does not tell how to distribute the clients.
  - Almost never used.
  - Does not add much value to fat clients.
  - Developers must learn a new product.
  - Little support.

The Client Layer, Cont'd

- Mobile phones and other hand held devices
  - Either browser or fat client.
  - From the view of the Java EE application they are not different from browsers or fat clients running on ordinary computers.

The Client Layer, Cont'd

- Clients using web services
  - Java EE supports servlets and EJBs as endpoints.
  - Java EE supports ordinary objects, servlets and EJBs as clients.
  - No need to write code for SOAP, UDDI, WSDL or anything like that.
  - Web services are a way to do RPC.
  - Not object oriented.
  - Widespread standard.

The Client Layer, Cont'd

- Clients using web services
  - Format that is easy to handle (text over HTTP).
  - Good choice for communication with other web based applications.

What is a framework?

- Subsystem that handles infrastructure (non-functional requirements).
  - transactions, security, networking, persistence etc.
- Part of Java EE or third party (or in-house).
- Callbacks, not just api.

Framework advantages

- Often used frameworks are thoroughly tested and proven to work well.
- Lots of documentation, easy to get help.
- New code means new bugs.

Framework advantages, Cont'd

- Non-functional requirements are difficult to code.
- Callback style makes sure all calls to non-functional requirements code are made at the right time.
  - Handled by the framework.

Frameworks Covered Here

- Presentation layer:
  - Struts, Spring, JSF
- Business logic layer:
  - Spring, EJB
- Integration layer:
  - Spring, JPA, Hibernate

Do not Write Infrastructure Code

- There are enough frameworks to handle all needs.
- Only write application specific code.
- No need to reinvent the wheel.
- It might take time to get started with a framework, but quite soon it saves time.

Which Framework Should We Use?

- That depends...
  - Developper knowledge.
  - How easy to use?
  - Will it be maintained in the future?
  - Compatible with previous, current and coming versions of java EE and other frameworks?
  - Does it meet our needs?
  - Customer requierements.
  - Frameworks used in existing applications.

Requirements on the Presentation Layer

- Navigation
  - What calls should be made to the model and which is the next view, provided the user has clicked YYY in view ZZZ.
  - The next view may differ depending on the outcome of the call to the model.
  - Answers to the above should be stated as a set of navigation rules.
  - Navigation rules must be easy to understand and to modify.

Requirements on the Presentation Layer, Cont'd

- Validation
  - Control of data entered by the user.
  - The presentation layer does not know the meaning of the data. It can only check for example that a certain field is a date or contains only digits. It is up to the model to check things that require business logic or database calls, for example to match a user id and a password.

Requirements on the Presentation Layer, Cont'd

- Validation, Cont'd
  - If validation fails the same view should be shown again together with error messages explaining why the validation failed.
  - Which validations are to be made on which fields and which error messages to show if they fail should be specified as a set of validation rules.
  - Validation rules should be easy to understand and to modify.

Requirements on the Presentation Layer, Cont'd

- Validation, Cont'd
  - It should be easy to reuse validation code since the same validations will probably be done on many different fields.
  - It is a good idea to use client side validation with Java Script since that does not require data to be sent to the server. Server side validation must also be active since the user may turn off Java Script.

Requirements on the Presentation Layer, Cont'd

- Flow control
  - To force the user to visit the views in a certain order.
  - Stop user from using the browser's reload, forward and back buttons and from using deep links.
  - For example only view the receipt directly after buying something, or not press reload if it takes a while to transfer money between bank accounts.

Requirements on the Presentation Layer, Cont'd

- Flow control, Cont'd
  - Do not confuse with navigation. Navigation tells in which order the user is allowed to visit the views, flow control stops the user from breaking this order.

Requirements on the Presentation Layer, Cont'd

- Composite views
  - Views often consist of several parts: header, footer, navigation menus, main content etc.
  - Many of these parts are common for many views.
  - In order to avoid duplicated code it must be possible to reuse both page fragments (html) and page layout (html tables or css).

Requirements on the Presentation Layer, Cont'd

- Internationalization (i18n) and localization (l10n)
  - Internationalization means to make it possible to switch language. To add the possibility to show the user interface in a new language should only require to write the words in the new language, not any additional coding.
  - Localization means to add support for a new language. This is quite easy, it is internationalization that is the tricky part.

Requirements on the Presentation Layer, Cont'd

- Session management
  - A session starts when the user accesses the site the first time and ends when the server decides to end it or when the user closes the browser.
  - Since Http is stateless it is necessary to use sessions to identify that two requests come from the same user.
  - The presentation layer must decide when sessions start and stop, how to associate users with sessions and how to associate data with sessions.

Requirements on the Presentation Layer, Cont'd

- Security
  - The presentation layer must handle authentication, authorization, logging and secure communication with the browser.

Requirements on the Presentation Layer, Cont'd

- Error handling
  - Which component is responsible for showing error messages?
  - Which component is responsible for logging error messages?
  - How do information about errors arrive to those components?
  - What information shall be logged and what information shall be shown to the user?

Requirements on the Presentation Layer, Cont'd

- Caching
  - Lots of time can be saved with good use of caches. It is probably not necessary neither to call the database nor to generate html dynamically for every call.
  - The most powerful cache is the browser's since it can be reached by the user without any interprocess call at all. We could for example cache all non-html pages (images, javascript, css etc) one hour in the browser and all html pages one minute.

Requirements on the Presentation Layer, Cont'd

- Caching, Cont'd
  - Another type of cache is a web cache, a separate product somewhere between the user and the server.
  - It takes longer for the user to reach but can serve more than one user. Such cashes are good for dynamic pages that are updated seldom, for example news sites.

Requirements on the Presentation Layer, Cont'd

- Caching, Cont'd
  - Yet another idea is to put an ordinary web server in front of the Java EE application server.
  - It is not a good idea to use a web server only as a cache, in that case a web cache is better.
  - It might be good to use a web server to serve static pages. The question is if the web server is so fast and the number of static pages so high that we gain the cost of the extra interprocess call from the web server to the Java EE application server.

Security

- Logging is done by the server and is thus server specific. It is not part of Java EE or any framework.
  - Well known log APIs are log4j from apache and the logging API in the JDK, often referred to as jdk 1.4 logging as it exists from that version.
  - These only handles how to write to the logs.
  - What and when to log is up to the server and the application code.

Security, Cont'd

- Authentication, authorization, privacy and integrity are handled by Java EE itself, so there is no need for a framework to do that.

Security, privacy and integrity, Cont'd

- Privacy and integrity are achieved by using HTTPS for communication.
- It can be specified in **`web.xml`** that all access to certain URLs should be with HTTPS. How that is implemented is up to the server, the developer need only worry about the lines in **`web.xml`**.

Security, privacy and integrity, Cont'd

- Specifying that HTTPS should be used:

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/some/url/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

  - The URL for which HTTPS shall be used.
  - The HTTP methods for which HTTPS shall be used. HTTP methods not listed here are not allowed at all.

Security, privacy and integrity, Cont'd

- Specifying that HTTPS should be used:

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/some/url/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

  - Means to use HTTPS instead of plain HTTP

Security, authentication

- There are many ways to log in to a web application, the most common is called form based login.
- It is specified in this way in **`web.xml`**:

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/login.html</form-login-page>
    <form-error-page>/login.html</form-error-page>
  </form-login-config>
</login-config>
```

- The URL of the page with the login form
- The URL of the page to be shown if login fails.

Security, authentication, Cont'd

- The name of the user id field in the login form must be **`j_username`**, the name of the password field must be **`j_password`** and the form action must be **`j_security_chec`**k.
- A form could look like this:

```
<form method="POST" action="j_security_check">
     <input type="text" name="j_username">
     <input type="password" name="j_password">
</form>
```

Security, authentication, Cont'd

- Always use HTTPS when the login form is presented. Otherwise the password will be transmitted in plain text.

Security, authorization

- Authorization is specified in **web.xml** by telling which user roles are allowed access to specified URLs.
- If a user has not logged in the container presents a login page.
  - The login page is the form specified above if form based login is used.

Security, authorization, Cont'd

- In **`web.xml`**:

```
  <web-resource-collection>
    <web-resource-name>Bank customer resorce</web-resource-name>
    <url-pattern>/some/url/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>customer</role-name>
  </auth-constraint>
</security-constraint>
```

- The URL of the pages with restricted access
- The HTTP methods for which access is restricted. HTTP methods not specified here are not allowed at all.
- Only users in this role are allowed access

Security, authorization, Cont'd

- User ids, passwords and roles are checked by *login modules*, as specified in the Java Authentication and Authorization Service (JAAS).
- How to install new login modules and add new users is not covered by any specification, it is up to the server.

Security, authorization, Cont'd

- How to handle login modules in Tomcat is specified in the Tomcat documentation, see **`http://tomcat.apache.org/tomcat-7.0-doc/realm-howto.html`**
- In GlassFish login modules are managed in the administration console, which is available on port 4848 when the server is running.

Frameworks for the presentation layer

- Struts (Apache)
- JSF (Sun, part of Java EE)
- Spring (VMware)
- And lots of other

Struts

Homepage:
http://struts.apache.org/

Documenation:
http://struts.apache.org/2.2.1.1/docs/home.html

Description of the architecture:
http://struts.apache.org/2.2.1.1/docs/big-picture.html

Struts, navigation

- Configured in the `struts.xml` file in `WEB-INF`

```
<action name="HelloWorld" class="tutorial.HelloWorld">
    <result>/HelloWorld.jsp</result>
</action>
```

  - The logical name of the action. This action is reached with the url **`http://url.for.my.server/HelloWorld.action`**. All requests pass through the **`FilterDispatcher`** filter which instantiates actions and calls them.
  - The class name of the `Action` class. Its method **`execute()`** is called if nothing else is specified.
  - Actions are automatically populated with form data.

Struts, navigation, Cont'd

- Configured in the struts.xml file in WEB-INF

```
<action name="HelloWorld" class="tutorial.HelloWorld">
    <result>/HelloWorld.jsp</result>
</action>
```

  - The next view. After the call to **`HelloWord.execute()`** returns, the http request is forwarded to **`/HelloWorld.jsp`**. It is possible to define different views depending on the outcome of the call to **`execute()`**.

Struts, validation

- Configured in separate files for each action, named after the action. To validate input in a request for **`HelloWorld.action`** the file is called **`HelloWorld-validation.xml`**
- Validators are called by an interceptor that is run before invoking the action, see the architecture description above.

Struts, validation, Cont'd

```
<validators>
    <field name="username">
        <field-validator type="requiredstring">
            <message>Username is required</message>
        </field-validator>
    </field>
</validators>
```

- The name of the http request parameter, that is the name of the field in the html form.
- The name of the validator.
- The error message if the validation fails.

Flow Control

- This is not handled by Struts.

Flow Control, Cont'd

- A way to code it is to always save a unique value, for example a random long, in the session object and also send the value to the client (cookie, hidden field or URL rewriting). When a request arrives the client and server values are compared and if they do not match the client has not followed the correct flow.

Flow Control, Cont'd

- Remember to decide what shall happen if the client has not followed the flow.

Struts, Composite Views

- This is not a core Struts functionality but there is another Apache project called *Tiles* that handles the problem very well,
  see **`http://tiles.apache.org/`**.

Struts, Internationalization

- There is good support in Java SE itself for internationalization, see **`http://download.oracle.com/javase/6/docs/technotes/guides/intl/index.html`**
- Struts internationalization support is described in the following guide:
  - **`http://struts.apache.org/2.2.1.1/docs/localization.html`**

Struts, Error Handling

- Struts can show different views dependent on the outcome of an action. This is specified in **`struts.xml`**:

```
<action name="MyAction" class="example.MyAction">
  <result name="result1">/somejsp.jsp</result>
  <result name="result2">/anotherjsp.jsp</result>
</action>
```

- The name of the actual outcome is specified by the action as a **`String`** return value from the **`execute`** method.

Java EE, Error handling

- It is possible to define error pages. If an exception occurs in a jsp or servlet the container forwards the call to the error page.
  - Error pages are defined like this in the **`web.xml`** deployment descriptor:

```
<!-- An error page for a Java exception. The call is
forwarded to the error page if the specified exception
or a subclass of it is thrown. -->
<error-page>
  <exception-type>java.lang.Throwable</exception-type>
  <location>/errorpage.jsp</location>
</error-page>

<!-- An error page for an HTTP error -->
<error-page>
  <error-code>404</error-code>
  <location>/errorpage.jsp</location>
</error-page>
```

JavaServer Faces, JSF

`javax.faces`

JSF Home page:

http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

JSF tag library documentation:

http://download.oracle.com/javaee/6/javaserverfaces/2.0/docs/pdldocs/facelets/

JSF, Navigation

- Static Navigation
  - If the outcome is the name of a XHTML page then that page is displayed.
  - This is called *static navigation*. The outcome is always the same.

JSF, Navigation, Cont'd

- Dynamic Navigation
  - A user action can often have different outcomes, for example a login attempt might succeed or fail.
  - In this case *dynamic navigation* must be used.

JSF, Dynamic Navigation, Cont'd

- Using dynamic navigation the value of the action attribute must be an expression identifying a method, for example **`#{loginManager.validateUser}`**, assuming that there is a managed bean named **`loginManager`** that has a method called **`validateUser`**.
- The outcome will be the value that is returned by this method. If the return value is not a **`String`** it will be converted to a **`String`** by calling its **`toString`** method.
- The outcome could be the name of a XHTML page, just like with static navigation. If so this page will be displayed.

JSF, Dynamic Navigation, Cont'd

- It is not a good design that methods in the model knows names of XHTML files.
- Therefore we want to have the action handling method return a logical view name that is mapped to a XHTML file name.

JSF, Dynamic Navigation, Cont'd

- This is achieved by adding a navigation rule to the `faces-config.xml` file

```
<navigation-rule>
    <from-view-id>/login.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/welcome.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>failure</from-outcome>
        <to-view-id>/login.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

- The above means that if an action handling method specified on the **`login.xhtml`** page returns **`success`** the **`welcome.xhtml`** page is displayed next. If on the other hand the method returns **`failure`** the **`login.xhtml`** page is displayed again.

JSF, Dynamic Navigation, Cont'd

- Even though the action handling method now returns a logical outcome, one could argue that we still have some amount of mixture of business logic and view handling.
- Consider for example a method **`withdraw`** in a bank application. Such a method would normally be **`void`**, but would now instead have to return the string **`success`** only to indicate to JSF that the withdrawal was successful.

JSF, Dynamic Navigation, Cont'd

- To avoid this problem we can let the **`withdraw`** method remain **`void`**, and instead add another method, **`success`**, that returns **`true`** only if the last transaction was successful. **`faces-config.xml`** would then look as follows.

```
<navigation-rule>
    <from-view-id>/withdraw.xhtml</from-view-id>
    <navigation-case>
        <if>#{bankManager.success}</if>
        <to-view-id>/success.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

JSF, Naviation, Cont'd

- No Matching Navigation Case
  - If there is an outcome that does not correspond to a XHTML file and that has no matching navigation case, the last page is displayed again.

JSF, Validation Example

```
<h:input id="name" label="Name"
         value="#{user.name}">
    <f:validateRequired/>
</h:inputText>
<h:message for="name"/>
```

- The **`validateRequired`** tag checks that the text field is not empty.
- The **`message`** tag displays the error message if the validation failed.
- It is possible to customize the error message.

JSF, Validation Cont'd

- When the *Bean Validation Framework* is used it is better not to use the JSF validation described on the previous slide.
  - If both Bean validation and JSF validations are used it becomes necessary to maintain two sets of validators, which means duplicated code.

JSF, Composite Views

- JSF, like Tiles, contains a lot of support for composite views.
- An easy way to handle composite views is to write a template XHTML file to define the layout, see next slide.
  - The **`ui:insert`** tag defines where different parts of the page are inserted.
  - The **`ui:include`** tag defines default parts.

JSF, Composite Views Example, Layout Template

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">

  <h:head>
    <title><ui:insert name="windowTitle"/></title>
  </h:head>

  <h:body>
    <ui:insert name="heading">
      <ui:include src="/default-header.xhtml"/>
    </ui:insert>

    <ui:insert name="sidebarLeft">
      <ui:include src="/default-sidebarLeft.xhtml"/>
    </ui:insert>

    <ui:insert name="content"/>
  </h:body>
</html>
```

JSF, Composite Views

- The views specifies which template to use and which page parts to insert at the **`ui:insert`** tags, see next slide..
  - The **`ui:composition`** tag defines which template to use.
  - The **`ui:define`** tag defines which parts to include.

JSF, Composite Views Example, Layout Template

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">

  <head><title>IGNORED</title></head>

  <body>
    <ui:composition template="/templates/masterLayout.xhtml">
      <ui:define name="windowTitle">
        The page title
      </ui:define>

      <ui:define name="content">
        The main content of the page
      </ui:define>
    </ui:composition>
  </body>
</html>
```

JSF, Internationalization

- As a simple example, content of a web page can be internationalized like this:
  1. Load the resource bundle with the **`f:loadBundle`** tag. The bundle is stored as a request attribute in a map:
     ```
     <f:loadBundle basename="example.MyMessages"
                   var="MyMessages"/>
     ```
     - **`example.MyMessages`** is the name of the resource bundle
     - **`MyMessages`** is the name of the map stored in the request context.
  2. Access the content of the bundle map with EL:
     `#{MyMessages.SomeMessageInTheBundle}`

JavaServer Faces, Flow Control

- Flow control is not handled by JSF.

JavaServer Faces, Error Handling

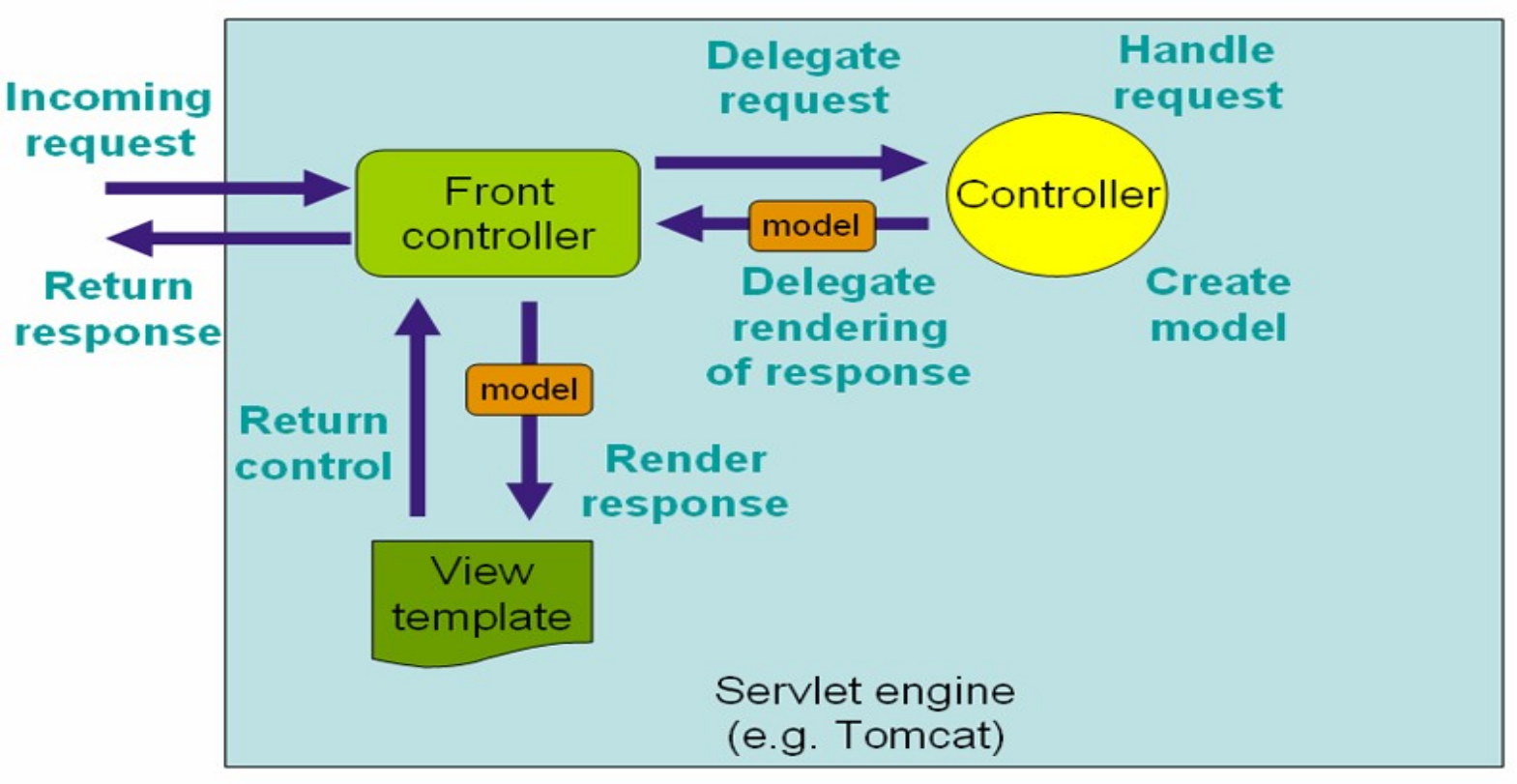
- It is possible to specify different views depending on the outcome of an operation, see slides concerning *navigation*.

Spring

- Home page, **`http://www.springsource.org/`**
- Very common for the business layer but less common for the presentation layer.

Spring, Architecture

- Controller servlet like Struts and JSF.



Spring, Architecture, Cont'd

- Based on the JavaBeans standard. All objects are beans.
- Dependency injection, very similar to CDI.
  - All beans can be configured in an xml configuration file or with annotations.
  - The beans, their relations and the initial values of their properties are configured.
  - This means that all relations between objects is already in place when they are first used.
  - It also solves the problem about initialization parameters to singletons.

Spring, Architecture, Cont'd

- Integrates very well with other frameworks.
- Contains no view technology but integrates easily with a lot of others, for example Struts Tiles, plain JSP or XSLT.

A comparison of presentation layer frameworks

- Struts:
  Very widespread, probably because it came years before the others. Relatively easy to use but has grown bigger and thus more complicated.
- JSF:
  Part of Java EE. Lots of improvements in version 2.0, which is notably easier to use and contains more functionality then older versions. Has gained a lot of popularity recently because of this.

A comparison of presentation layer frameworks, Cont'd

- Spring:
  - Widely used, especially in the business layer.
  - Quite easy to use relative to its richness.
  - Integrates very easily with other frameworks (for example Struts and JSF) so the fact that it is used in the business layer is not an argument to use it in the presentation layer.