Modulo I
Frameworks Web

Prof. Ismael H F Santos

Bibliografia

- Spring in Action
  - Craig Walls and Ryan Breidenbach
- Professional Java Development with Spring
  - Rod Johnson, Juergen Hoeller and Team

Ementa

- Modelos MVC (Model View Controller) e MVC2
- Application Layering
- Comparação Frameworks Web

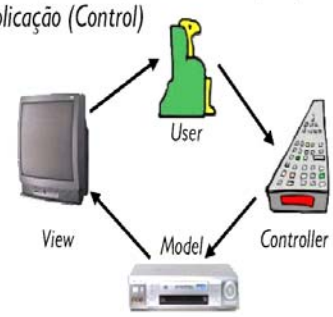
WebApp

Modelos MVC e MVC2

Arquitetura MVC

- Surgiu nos anos 80 com a linguagem SmallTalk
- Divide a aplicação em tres partes fundamentais
  - Model – Representa os dados da aplicação e as regras de negócio (business logic)
  - View – Representa a informação recebida e enviada ao usuário
  - Controller – Recebe as informações da entrada e controla o fluxo da aplicação

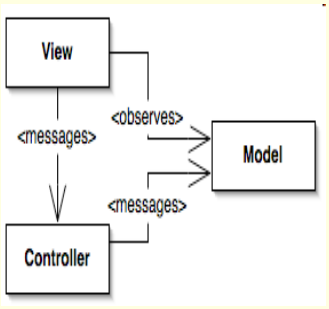
- *Técnica para separar dados ou lógica de negócios (Model) da interface do usuário (View) e do fluxo da aplicação (Control)*



Fonte: http://www.computer-programmer.org/articles/struts/

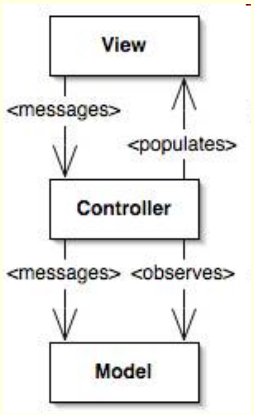
MVC original

- O padrão de arquitetura MVC (model-view-controller) surgiu na comunidade smalltalk.
  - Criado por Trygve Reenskaug no fim dos anos 70
  - Usado no desenvolvimento de aplicações desktop por facilitar o desenvolvimento em camadas de aplicações que usam a orientação a objetos



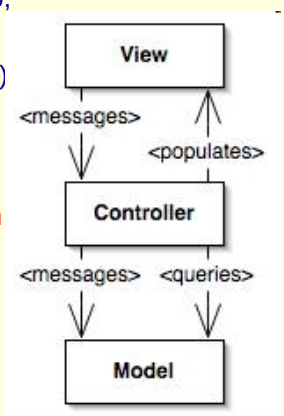
MVC Next – Steve Jobs

- A next (Steve Jobs), resolveu modificar esse modelo oferecendo uma alternativa para sua linguagem de programação objective-c.
  - Delega a responsabilidade de observar o modelo para a camada de Controller que, por sua vez, envia para a camada de visão as alterações ao invés da camada de View obter esses dados do Model.



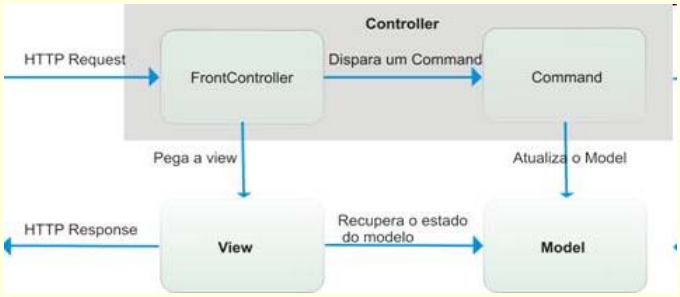
MVC Model 2

- Com o crescimento das aplicações web baseadas no protocolo HTTP que é sem estado, não temos mais uma sessão permanentemente aberta entre o cliente e o servidor. Além disso o HTTP não prevê uma forma de "enviar" (push) informações do servidor para o cliente.
- Isto impede o trabalho do Controller que não pode mais enviar informações para a View sem ser solicitado. Para contornar o problema a Sun criou o MVC Model 2, baseado no padrão FrontController.
- Agora a camada Controller submete ações tentando acompanhar o processo de request-response do protocolo HTTP ao invés de observar a camada Model, criando um fluxo linear para a arquitetura das aplicações.



Padrão Front Controller

- Padrão que consolida todas as requisições web em um único objeto manipulador, despachando o tratamento adequado dessas requisições conforme o comportamento esperado.



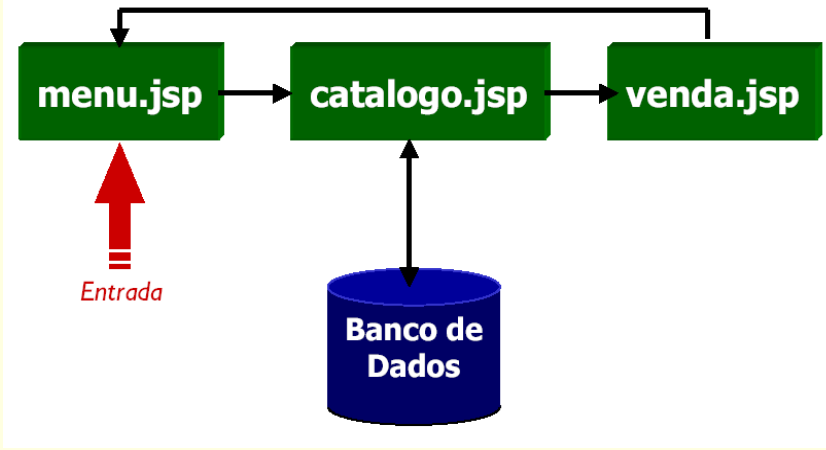
- A seguir apresentamos mais detalhes do padrão

Implementação do MVC para Web

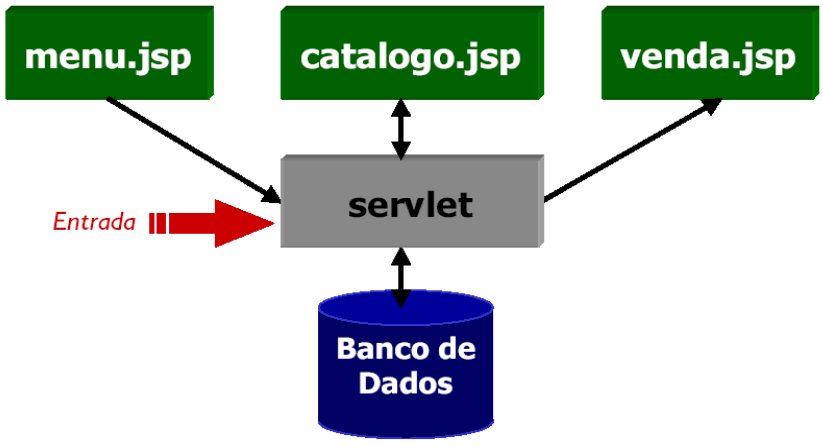
- *Design centrado em páginas*
  - *Aplicação JSP consiste de seqüência de páginas (com ou sem beans de dados) que contém código ou links para chamar outras páginas*
- *Design centrado em servlet (FrontController* ou MVC)*
  - *Aplicação JSP consiste de páginas, beans e servlets que controlam todo o fluxo de informações e navegação*
  - *Este modelo favorece uma melhor organização em camadas da aplicação, facilitando a manutenção e promovendo o reuso de componentes.*
  - *Um único servlet pode servir de fachada*
  - *Permite ampla utilização de J2EE design patterns*

** FrontController é um J2EE design pattern. Vários outros design patterns serão identificados durante esta secão. Para mais informacões, veja Sun Bluebrints [7]*

JSP Model I - Centrado em páginas



JSP Model II - Centrado em servlet

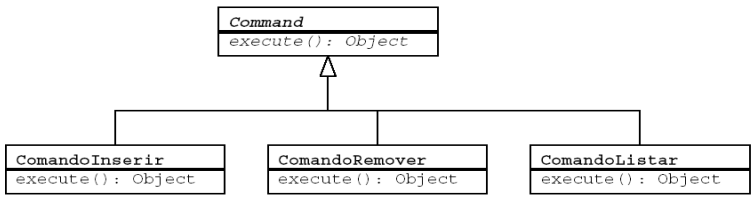


Como implementar ?

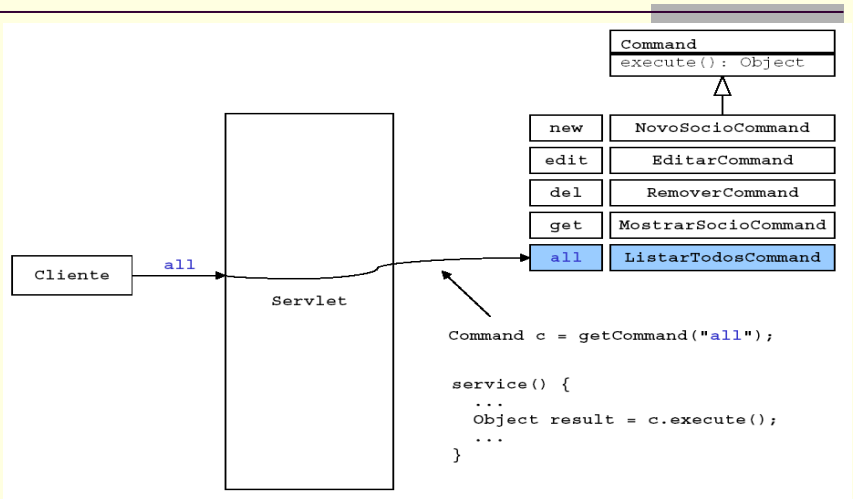
- *Há várias estratégias*
- *Todas procuram isolar*
  - *As operações de controle de requisições em servlets e classes ajudantes,*
  - *Operações de geração de páginas em JSP e JavaBeans, e*
  - *Lógica das aplicações em classes que não usam os pacotes javax.servlet*
- *Uma estratégia consiste em se ter um único controlador (FrontController pattern) que delega requisições a diferentes objetos que implementam comandos que o sistema executa (Command pattern)*

Pattern Command (GoF)

- *É um padrão de projeto clássico catalogado no livro "Design Patterns" de Gamma et al (GoF = Gang of Four)*
  - *Para que serve: "Encapsular uma requisição como um objeto, permitindo que clientes parametrizem diferentes requisições, filas ou requisições de log, e suportar operações reversíveis." [GoF]*
- *Consiste em usar polimorfismo para construir objetos que encapsulam um comando e oferecer um único método execute() com a implementação do comando a ser executado*



Pattern Command (GoF)



FrontController + Command

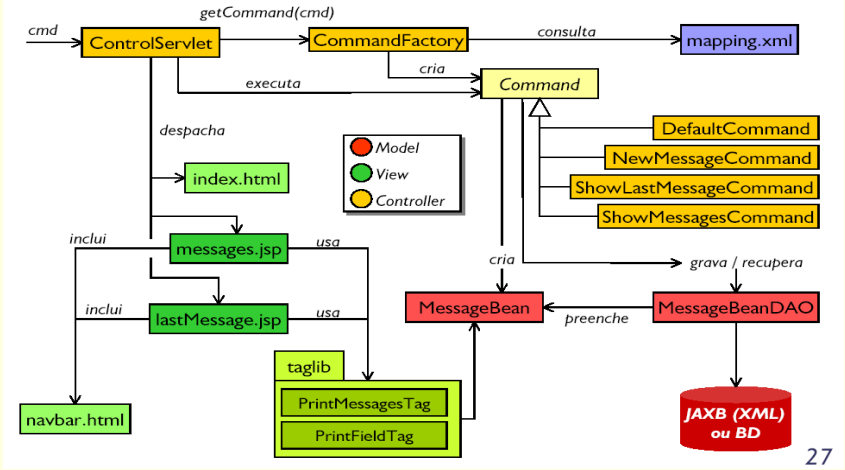
- *Os comandos são instanciados e guardados em uma base de dados na memória (HashMap, por exemplo)*
  - *Pode-se criar uma classe específica para ser fábrica de comandos*
- *O cliente que usa o comando (o servlet), recebe na requisição o nome do comando, consulta-o no HashMap, obtém a instância do objeto e chama seu método execute()*
  - *O cliente desconhece a classe concreta do comando. Sabe apenas a sua interface (que usa para fazer o cast ao obtê-lo do HashMap*
- *No HashMap*

```
Comando c = new ComandoInserir();
comandosMap.put("inserir", c);
```

- *No servlet:*

```
String cmd = request.getParameter("cmd");
Comando c = (Comando)comandosMap.get(cmd);
c.execute();
```

Exemplo de Implementação – hellojsp_2



Mapemamentos de comandos ou ações

- *No exemplo hellojsp_2, o mapeamento está armazendo em um arquivo XML (webinf/mapping.xml)*

```
<command-mapping> (...)
    <command>
        <name>default</name>
        <class>hello.jsp.DefaultCommand</class>
        <success-url>/index.html</success-url>
        <failure-url>/index.html</failure-url>
    </command>
    <command>
        <name>newMessage</name>
        <class>hello.jsp.NewMessageCommand</class>
        <success-url>/lastMessage.jsp</success-url>
        <failure-url>/index.html</failure-url>
    </command>
    <command>
        <name>showAllMessages</name>
        <class>hello.jsp.ShowMessagesCommand</class>
        <success-url>/messages.jsp</success-url>
        <failure-url>/index.html</failure-url>
    </command>
</command-mapping>
```

Comandos ou ações (Service to Worker)

- *Comandos implementam a interface Command e seu método*

```
Object execute(HttpServletRequest request,
               HttpServletResponse response,
               MessageBeanDAO dao);
```

- *Criados por CommandFactory na inicialização e executados por ControlServlet que os obtém via getCommand(nome)*
- *Retornam página de sucesso ou falha (veja mapping.xml)*
- *Exemplo: ShowMessagesCommand:*

```
public class ShowMessagesCommand implements Command {

   public Object execute(...) throws CommandException {
        try {
            MessageBean[] beanArray = dao.retrieveAll();
            request.setAttribute("messages", beanArray);
            return successUrl;
        } catch (PersistenceException e) {
            throw new CommandException(e);
        }
   } (...)
```


Data Access Objects (DAO)

- *Isolam a camada de persistência*
  - *Implementamos persistência JAXB, mas outra pode ser utilizada (SGBDR) sem precisar mexer nos comandos.*
- *Interface da DAO:*

```
public interface MessageBeanDAO {
   public Object getLocator();

   public void persist(MessageBean messageBean)
                            throws PersistenceException;

   public MessageBean retrieve(int key)
                            throws PersistenceException;

   public MessageBean[] retrieveAll()
                            throws PersistenceException;

   public MessageBean retrieveLast()
                            throws PersistenceException;
}
```

Controlador (FrontController)

- *Na nossa aplicação, o controlador é um servlet que recebe os nomes de comandos, executa os objetos que os implementam e repassam o controle para a página JSP ou HTML retornada.*

```
public void service( ..., ... ) ... {
    Command command = null;
    String commandName = request.getParameter("cmd");

    if (commandName == null) {
        command = commands.getCommand("default");
    } else {
        command = commands.getCommand(commandName);
    }

    Object result = command.execute(request, response, dao);
    if (result instanceof String) {
        RequestDispatcher dispatcher =
              request.getRequestDispatcher((String)result);
        dispatcher.forward(request, response);
    }
   ...
}
```

Método de CommandFactory

Execução do comando retorna uma URI

Repassa a requisição para página retornada

ValueBean ViewHelper (Model)

- *Este bean é gerado em tempo de compilação a partir de um DTD (usando ferramentas do JAXB)*

```
public class MessageBean
     extends MarshallableRootElement
     implements RootElement {

     private String _Time;
     private String _Host;
     private String _Message;

     public String getTime() {...}
     public void setTime(String _Time) {...}

     public String getHost() {...}
     public void setHost(String _Host) {...}

     public String getMessage() {...}
     public void setMessage(String _Message) {...}

     ...
}
```

interfaces JAXB permitem que este bean seja gravado em XML (implementa métodos marshal() e unmarshal() do JAXB)

Pagina JSP (View) com custom tags

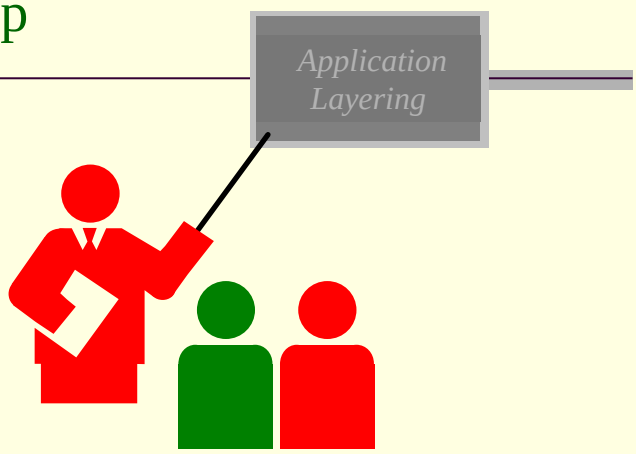
- *Página messages.jsp (mostra várias mensagens)*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<%@ taglib uri="/hellotags" prefix="hello" %>
<html>
<head><title>Show All Messages</title></head>
<body>
<jsp:include page="navbar.html" />
<h1>Messages sent so far</h1>
<table border="1">
<tr><th>Time Sent</th><th>Host</th><th>Message</th></tr>

<hello:printMessages array="messages">
    <tr>
        <td><hello:printField property="time" /></td>
        <td><hello:printField property="host" /></td>
        <td><hello:printField property="message" /></td>
    </tr>
</hello:printMessages>

</table>
</body>
</html>
```


WebApp



Spring

Application Layering

- A clear separation of application component responsibility.
  - **Presentation layer**
    - Concentrates on request/response actions
    - Handles UI rendering from a model.
    - Contains formatting logic and non-business related validation logic.
    - Handles exceptions thrown from other layers
  - **Persistence layer**
    - Used to communicate with a persistence store such as a relational DB
    - Provides a query language
    - Possible O/R mapping capabilities
      - JDBC, Hibernate, iBATIS, JDO, Entity Beans, etc.
  - **Domain layer**
    - Contains business objects that are used across above layers.
    - Contain complex relationships between other domain objects
    - May be rich in business logic
    - May have ORM mappings
    - Domain objects should only have dependencies on other domain objs

Spring

Application Layering (cont)

- Where do we code business logic?
  - **Domain objects**
    - Heavy domain model / thin service layer approach
    - Business logic is embedded in domain objects
    - Takes advantage of OO programming
    - Behavior rich domain model
  - **Service Layer**
    - Thin domain model / heavy service layer approach
    - Wraps procedural business logic over domain objects
    - Anti-pattern according to Fowler – 'Anemic Domain Model'
    - Provides a separation of business logic concerns from the domain model
    - Treats the domain model as ORM objects

Spring

Application Layering (cont)

- More Architectural decisions…
  - How do we achieve independent layers of code that can provide clear separation, loose coupling, and allow communication with each other?
  - How can we design an architecture that can allow layer replacement without affecting existing layers?
  - What technologies, and frameworks, should be implemented in each layer?
  - How will we implement security?
  - Will the application be flexible to technology changes?
  - How will the application handle enterprise level services such as transactions, security, logging, resource pooling, profiling, etc?

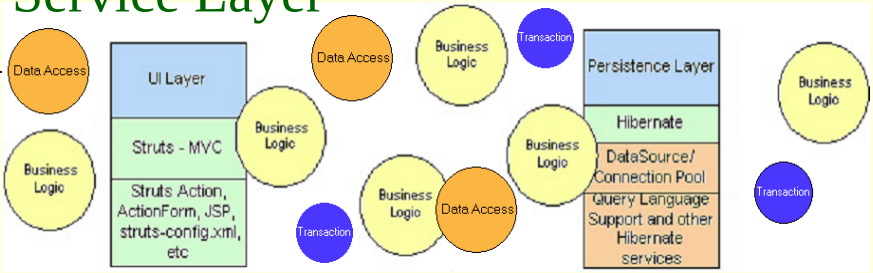
Spring

Application Layering (cont)

- Service layer
  - Gateway to expose business logic to the outside world
  - Manages ‘container level services’ such as transactions, security, data access logic, and manipulates domain objects
  - Not well defined in many applications today or tightly coupled in an inappropriate layer.

Spring

Service Layer



- Where do we position loosely-coupled business logic? What is service logic?
- How should container level services be implemented?
- How do we support transactions in a POJO based application?
- How do we communicate from our presentation layer to our persistence layer?
- How do we get to services that contain business logic?
- How should our business objects communicate with our persistence layer?
- How do we get objects retrieved from our persistence layer to our UI layer?

Spring

More about the Service Layer

- Often tightly coupled with other layers
  - Struts is not where you place business logic and persistence logic!
- The missing link IMHO in most applications today.
- EJB – SLSB, SFSB provide the common J2EE business layer enterprise solutions for transactions within a container. What about POJO?
- Hand code transaction logic with JTA
- Frameworks – Spring, Picocontainer, HiveMind, etc.
- Lighterweight containers use
  - IoC/Dependency Injection
  - AOP

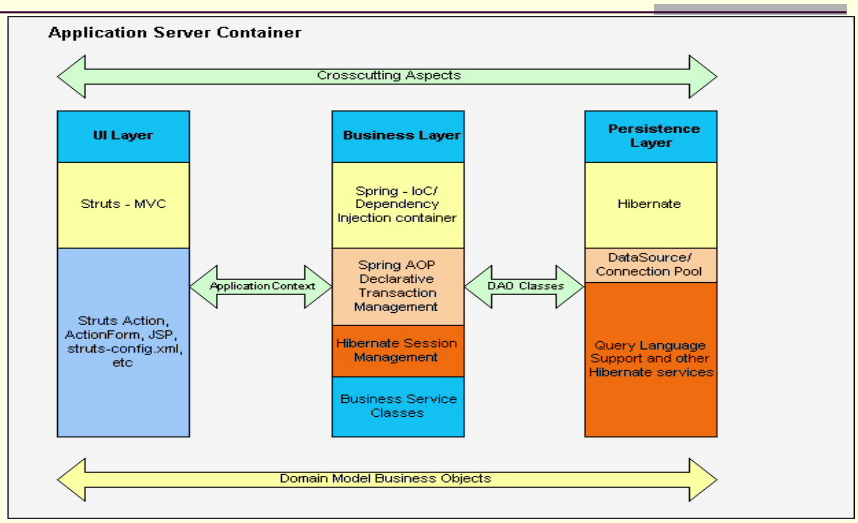
Spring

Real World EJB Usage

- Stateless Session Beans (SLSBs)
  - One of the easiest beans to program, but still want to program with POJOs.
  - Local interfaces alleviate remote interface performance issues.
  - Used as facades to other objects
  - Allows developers to provide declarative transaction management.
- Message Driven Beans (MDBs)
  - Easy to program
  - Provides asynchronous messaging
- Distributed Transaction Management
- RMI Remoting
- How do we get the benefits of EJB without using an EJB container?

Spring

Proposed Web App Layering

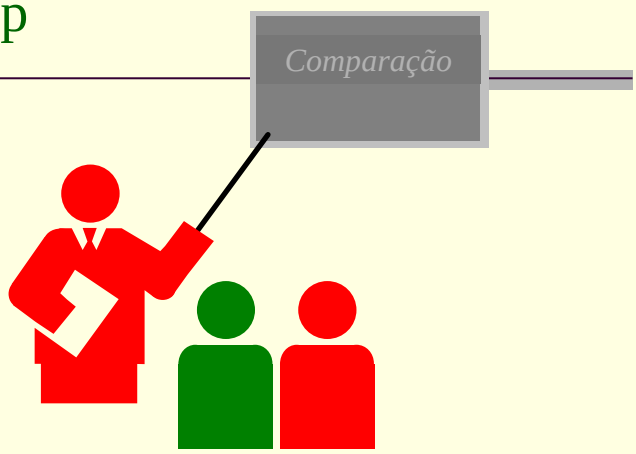


Spring

More Application Layering Combinations

- Presentation/Business/Persistence
  - Struts+Spring+Hibernate
  - Struts + Spring + EJB
  - JavaServer Faces + Spring + iBATIS
  - Spring + Spring + JDO
  - Flex + Spring + Hibernate
  - Struts + Spring + JDBC
- You decide…

WebApp



Comparação entre Framewoks Web

- Struts - http://struts.apache.org
  - Pros:
    - The “Standard” - lots of Struts jobs
    - Lots of information and examples
    - HTML tag library is one of the best
  - Cons:
    - ActionForms - they’re a pain
    - Can’t unit test - StrutsTestCase only does integration
    - Mailing list volume is overwhelming

Comparação entre Framewoks Web

- Spring MVC - http://www.springframework.org
  - Pros:
    - Lifecyle for overriding binding, validation, etc.
    - Integrates with many view options seamlessly: JSP/JSTL,Tiles, Velocity, FreeMarker, Excel, XSL, PDF
    - Inversion of Control makes it easy to test
  - Cons:
    - Not many using it
    - Requires writing lots of code in JSPs
    - Almost too flexible - no common parent Controller

Comparação entre Framewoks Web

- WebWork - http://opensymphony.org/webwork
  - Pros:
    - Simple architecture - easy to extend
    - Tag Library is easy to customize - backed by Velocity
    - Interceptors are pretty slick
  - Cons:
    - Documentation only recently written, few examples
    - Client-side validation immature

Comparação entre Framewoks Web

- Tapestry - http://jakarta.apache.org/tapestry
  - Pros:
    - Very productive once you learn it
    - Templates are HTML - great for designers
    - Healthy and smart user community
  - Cons:
    - Documentation very conceptual, rather than pragmatic
    - Steep learning curve - very few examples
    - Impossible to test - page classes are abstract

Comparação entre Framewoks Web

- JSF - http://java.sun.com/j2ee/javaserverfaces & http://myfaces.org
  - Pros:
    - J2EE Standard - lots of demand and jobs
    - Fast and easy to develop with Rich Navigation framework
  - Cons:
    - Tag soup for JSPs
    - Immature technology - doesn't come with everything
    - No single source for implementation

Controllers and Views

- Struts:
  - UserAction extends DispatchAction
- Spring MVC:
  - UserFormController extends SimpleFormController
- WebWork:
  - UserAction extends ActionSupport
- Tapestry:
  - UserForm extends BasePage
- JSF:
  - UserForm

List Screens

- How easy is it to integrate a sortable/pageable list of data?
  - Struts, Spring MVC and WebWork
    - all use Tag Libraries like the Display Tag
  - Tapestry has a contrib:
    - Table component
  - JSF
    - has a dataTable with no sorting – have to write your own logic if you want it

Bookmarking and URLs

- Using container-managed authentication (or other filterbased security systems) allow users to bookmark pages. They can click the bookmark, login and go directly to the page.
- WebWork
  - has namespaces - makes it easy
- Struts and Spring
  - allow full URL control
- Tapestry
  - has ugly URLs - difficult to segment the app for different roles
- JSF
  - does a POST for everything

Validation

- Validation should be easy to configure, be robust on the client side and either provide good out of the box messages or allow you to easily customize them.
- Struts and Spring MVC
  - use Commons Validator – a mature solution
- WebWork
  - uses OGNL for powerful expressions - client-side support very new
- Tapestry
  - has very robust validation - good messages without need to customize
- JSF
  - ugly default messages, but easiest to configure

Testability

- Struts
  - can use StrutsTestCase
- Spring and WebWork
  - allow easy testing with mocks (i.e. EasyMock, jMock, Spring Mocks)
- Tapestry
  - is impossible to test because page classes are abstract
- JSF
  - page classes can be easily tested and actually look a lot like WebWork actions

Success Messages

- The duplicate-post problem, what is it?
  - Easiest way to solve: redirect after POST
- Struts
  - is the only framework that allows success messages to live through a redirect
- Spring and WebWork
  - require custom solutions
- Tapestry
  - requires you to throw an Exception to redirect
- JSF
  - requires a custom solution, i18n messages difficult to get in page beans

Spring Integration

- All frameworks have integration with Spring
- Struts:
  - ContextLoaderPlugin and Base classes
- WebWork:
  - SpringObjectFactory
- Tapestry:
  - override base engine, grab from servlet context, put into global map
- JSF:
  - DelegateVariableResolver or JSF-Spring Library

Internationalization

- JSTL's <fmt:message> tag makes it easy
- No standard for getting i18n messages in controller classes
- Struts, Spring and JSF
  - encourage one ResourceBundle per locale
- WebWork and Tapestry
  - advocate separate files for each page/action

Page Decoration

- Used Tiles since it first came out in 2001
- SiteMesh is much easier to setup and use
- Tiles can be used in:
  - Struts, Spring and JSF
- Requires configuration for each page
- SiteMesh can be used with all frameworks
  - Requires very little maintenance after setup

Tools

- Struts
  - has a lot of IDE support and even has frameworks built on top of it (i.e. Beehive's PageFlow)
- Spring
  - has Spring IDE, only does XML validation, not a UI/web tool
- WebWork
  - has none
- Tapestry
  - has Spindle - great for coders
- JSF
  - has many, all cost money and hook into proprietary app servers