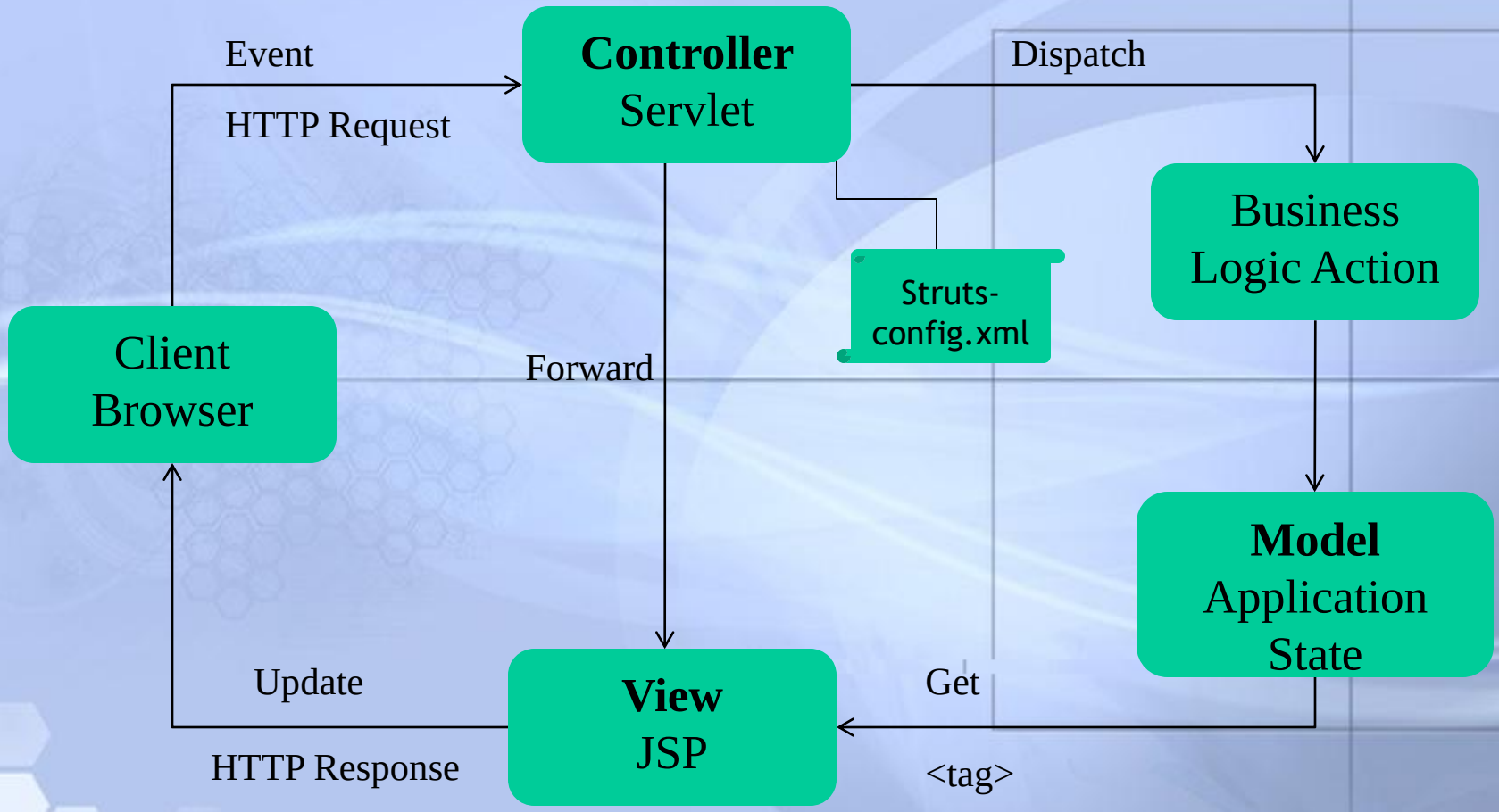
Session-11

Struts

Contents

- Struts Architecture
  - Model
  - View
  - Controller
- Components of Struts

Struts Architecture



Struts: MVC-based Architecture

- Central controller mediates application flow and delegates appropriate handlers called Action.
- Action handlers can use model components.
- Model encapsulates business logic or state.
- **Controller** forwarded back through the Controller to the appropriate view.
  - The forwarding can be determined by consulting a set of mappings in the configuration file, which provides a loose coupling between the **View** and **Model**.

Struts: MVC-based Architecture

- **Three major components in Struts:**

  1) Servlet controller (Controller)

  2) Java Server Pages or any other presentation technology (View)

  3) Application Business Logic in the form of whatever suits the application (Model)

- Struts is focused on **Controller**
  - Struts is Model and View independent
  - Struts can use any Model and View technologies

Struts: MVC-based Architecture

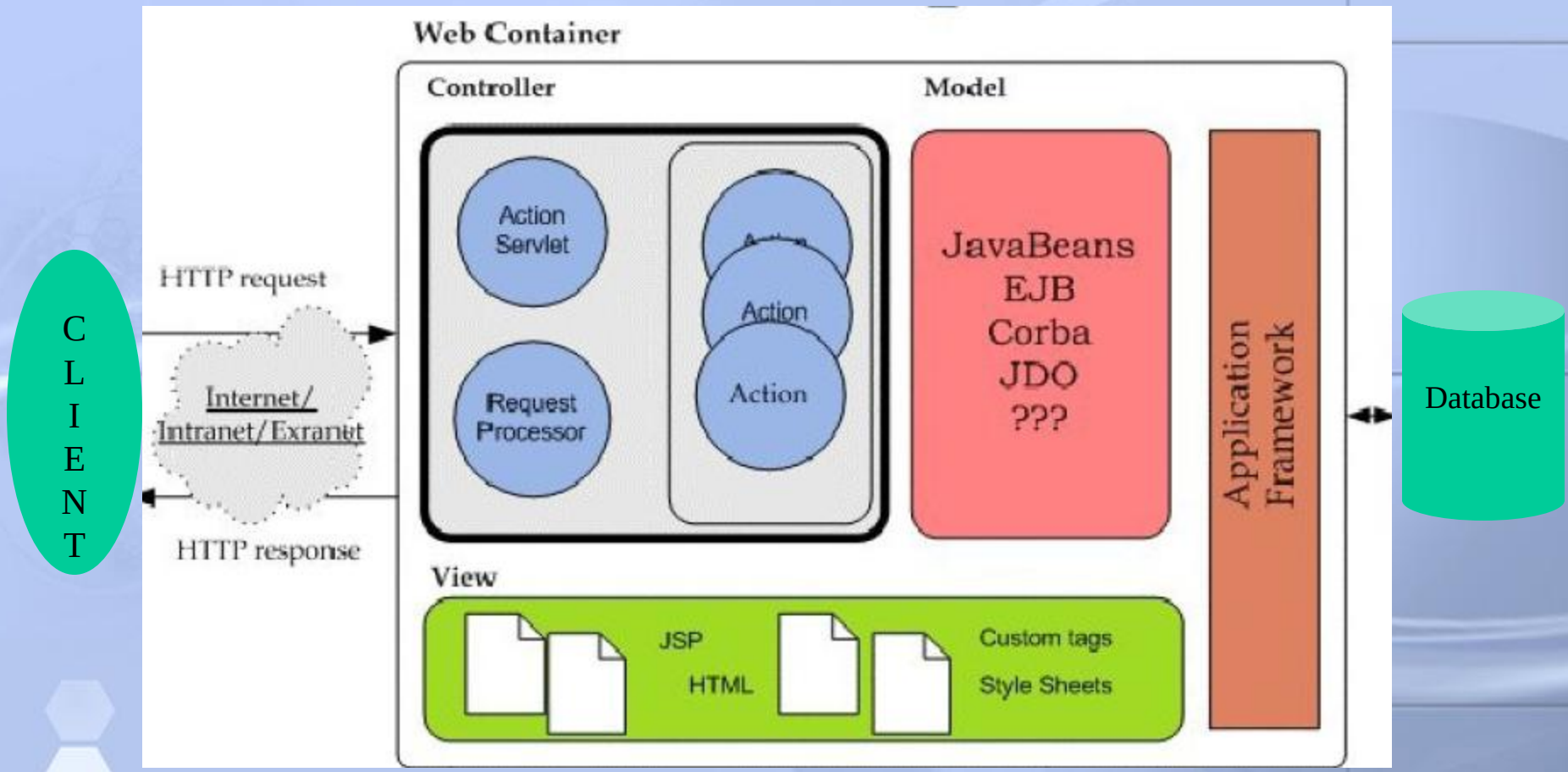
- **Configuration file contains action mappings**
  - URL to Action mappings
  - Controller uses these mappings to turn HTTP requests into application actions
  - Determines forwarding/navigation
- **Mapping must specify**
  - A request path
  - Action to act upon the request

Controller

What does Controller do?

- Is the **switch board** of MVC architecture.
- Every request goes through the controller.
- Responsible for flow control (action mapping) of the request handling
  - reads configuration file to determine the flow control

Controller Components



Controller in Struts Framework

- Struts framework provides a built-in base servlet
  - **org.apache.struts.action.ActionServlet**
  - Servlet mapping has to be configured in **web.xml**
- Struts related configuration is done through **struts-config.xml**
  - Action Mapping defines the mapping between request URI of incoming requests to specific Action class

Developer Responsibility

- Write an **Action** class (that is, an extension of the Action class) for each logical request that may be received
  - override **execute()** method (perform() method in Struts 1.0)
- Write the action mapping configuration file
  - **struts-config.xml**
- Update the web application deployment descriptor file to specify the **ActionServlet**
  - **web.xml**

Controller Components in Struts Framework

- **ActionServlet** (Provided by Struts)
- **RequestProcessor** (Struts 1.1)(Provided by Struts)
  - One for each module
- **Action** (Provided by developer)
  - Developer extends Struts-provided Action class
- **Action Mapping** (Specified by developer)
  - Developer specifies action mapping in struts-config.xml file
  - Struts framework creates ActionMapping object and passes it to Action object

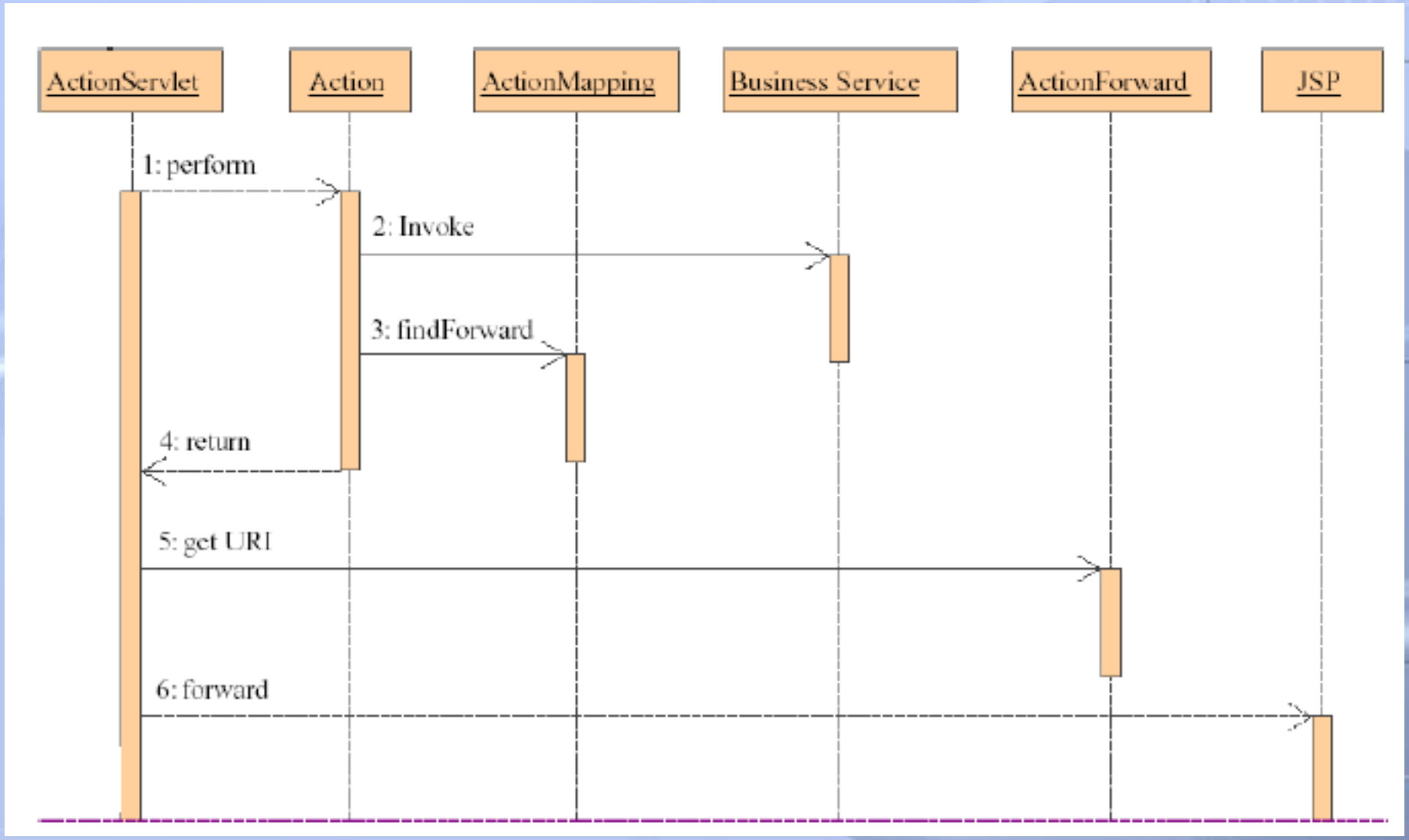
ActionServlet

- **Performs the role of Controller**
  - Process user requests
  - Determine what the user is trying to achieve according to the request
  - Pull data from the model (if necessary) to be given to the appropriate view, and
  - Select the proper view to respond to the user
- **Delegates most of this grunt work to Action classes**

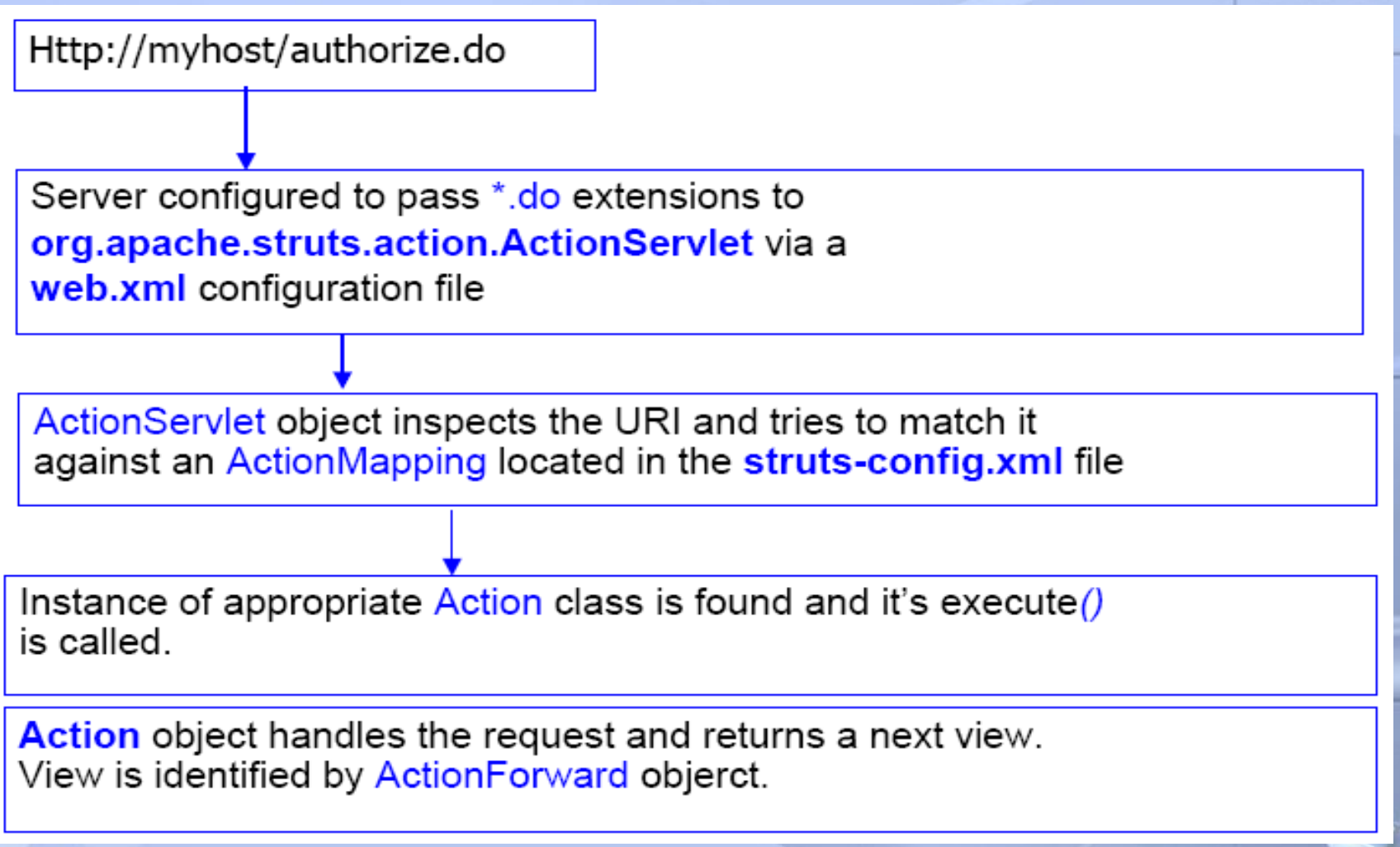
ActionServlet (Cont.)

- Is responsible for initialization and clean-up of resources
  - loads the application config corresponding to the **"config"** init-param's in **web.xml**
  - goes through an enumeration of all init-param elements, looking for those elements who's name starts with **config/** for modules
  - To access the module foo, you would use a URL like:
    - http://localhost:8080/myApp/foo/someAction.do

Flow control by Controller



Struts Flow


Http://myhost/authorize.do
Server configured to pass *.do extensions to
org.apache.struts.action.ActionServlet via a
web.xml configuration file
ActionServlet object inspects the URI and tries to match it
against an ActionMapping located in the struts-config.xml file
Instance of appropriate Action class is found and it's execute()
is called.
Action object handles the request and returns a next view.
View is identified by ActionForward objerct.

Request Processor

RequestProcessor

What Does RequestProcessor Do?

- **processPath**

  – Determine the path that invoked us. This will be used later to retrieve an ActionMapping.

- **processLocale**

  – Select a locale for this request, if one hasn‘t already been selected, and place it in the request.

- **processContent**

  – Set the default content type (with optional character encoding) for all responses if requested.

RequestProcessor

- **processNoCache**

  – If appropriate, set the following response headers: "Pragma", "Cache-Control", and "Expires".

- **processPreprocess**

  – This is one of the "hooks" the RequestProcessor makes available for subclasses to override. The default implementation simply returns true. If you subclass RequestProcessor and override processPreprocess you should either return true (indicating process should continue processing the request) or false (indicating you have handled the request and the process should return)

RequestProcessor

- **processMapping**

  – Determine the ActionMapping associated with this path.

- **processRoles**

  – If the mapping has a role associated with it, ensure the requesting user is has the specified role. If they do not, raise an error and stop processing of the request.

- **processActionForm**

  – Instantiate (if necessary) the ActionForm associated with this mapping (if any) and place it into the appropriate scope.

RequestProcessor

- **processPopulate**

  – Populate the ActionForm associated with this request, if any.

- **processValidate**

  – Perform validation (if requested) on the ActionForm associated with this request (if any).

- **processForward**

  – If this mapping represents a forward, forward to the path specified by the mapping.

RequestProcessor

- **processInclude**

  – If this mapping represents an include, include the result of invoking the path in this request.

- **processActionCreate**

  – Instantiate an instance of the class specified by the current ActionMapping (if necessary).

- **processActionPerform**

  – This is the point at which your action's perform() or execute() method will be called.

RequestProcessor

- **processForwardConfig**

  – Finally, the process method of the RequestProcessor takes the ActionForward returned by your Action class, and uses to select the next resource (if any). Most often the ActionForward leads to the presentation page that renders the response.

Action Mapping

Action Mapping in Struts Config File

- Struts controller **ActionServlet** needs to know several things about how each request **URI** should be **mapped** to an appropriate **Action** class.

- These requirements are encapsulated in a Java **interface** named **ActionMapping**.

  – Struts framework creates **ActionMapping** object from **<ActionMapping>** configuration element of struts-config.xml file.

Struts Config File (struts-config.xml)

- **struts-config.xml** contains three important elements used to describe actions:

  – **<form-beans>** contains **FormBean** definitions including **name** and **type** (classname)

  – **<action-mapping>** contains action definitions
    - Use an **<action>** element for each action defined

  – **<global-forwards>** contains your global forward definitions

Struts Config File (struts-config.xml)

- <form-beans>

  – This section contains your form bean definitions.

  – You use a <form-bean> element for each form bean, which has the following important attributes:

    - **name**: The name of the request (and session level attribute that this form bean will be stored as)
    - **type**: The fully-qualified Java classname of your form bean

struts-config.xml: <form-beans>

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3 <!DOCTYPE struts-config PUBLIC
4 "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
5 "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
6
7 <struts-config>
8
9 <!-- ========== Form Bean Definitions ================== -->
10 <form-beans>
11
12 <form-bean   name="submitForm"
13                   type="submit.SubmitForm"/>
14
15 </form-beans>
```

struts-config.xml: <action-mappings>

Each <action> element requires the following attributes to be defined:

- **path**: The application context-relative path to the action (URI of the request)
- **type**: The fully qualified java classname of your Action class
- **name**: The name of your <form-bean> element to use with this action
- **input**: The name of the display page when input form validation error condition occurs
- **scope**: The scope in which form bean is created
- **validate**: Whether to perform input form validation or not

struts-config.xml: <action-mappings>

```
1 <!-- ========== Action Mapping Definitions ============ -->
2 <action-mappings>
3
4     <action         path="/submit"
5                     type="submit.SubmitAction"
6                     name="submitForm"
7                     input="/submit.jsp"
8                     scope="request"
9                     validate="true">
10        <forward name="success" path="/submit.jsp"/>
11        <forward name="failure" path="/submit.jsp"/>
12    </action>
13
14 </action-mappings>
15
16 </struts-config>
```

Action Mapping Config File

<global-forwards>

- Forwards are instances of the ActionForward class returned from an Action's execute() method
- These map logical names to specific resources (typically JSPs)
- <forward> element has following attributes
  - **name**: logical name of the forward
    - used within execute() method of Action class to forward to the next resource
  - **path**: to-be-forwarded resource
  - **redirect**: redirect (true) or forward (false)

- Local forwarding can be done in <action-mapping> element.

struts-config.xml: < global-forwards >

```
<global-forwards
                type="org.apache.struts.action.ActionForward">
                <forward   name="logon“
                           path="/logon.jsp“
                           redirect="false" />
</global-forwards>
```

ActionForm

ActionForm Bean (Form bean)

- Provided by developer
  - Define an **ActionForm** bean (that is, a Java class extending the ActionForm class) for the input form
  - Define it in struts-config.xml file
    - **<form-bean>**
    - **name** attribute of **<Action>** class
- Contains only property getter and property setter methods for each field-no business logic
- Provides standard validation mechanism

ActionForm Bean & Controller

- For each **ActionForm** bean defined in **servlet-config.xml** file, Controller (ActionServlet) will
  - Check session scope for an instance of ActionForm bean
    - If it does not exist, controller creates one
  - Call corresponding setter method of ActionForm bean for every request parameter whose name corresponds to the name of a property of the bean
  - Pass the updated ActionForm bean object as a parameter to **execute()** method of Action class

How to write ActionForm Bean

- Add just getter and setter methods for each property of a input form.
  - Do not include any business logic code.
- Add a standard validation method called **validate()**
  - Controller will call this validation
- Define a property (with associated getXxx and setXxx methods) for each field that is present in the input form

Example submit.jsp

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<html>
 <head><title>Submit example</title></head>
 <body>
      <h3>Example Submit Page</h3>
      <html:errors/>
     <html:form action="submit.do">
            First Name: <html:text property="firstName"/><br>
            Last Name: <html:text property="lastName"/><br>
            <html:submit/>
       </html:form>
</body>
</html>
```

Model: ActionForm

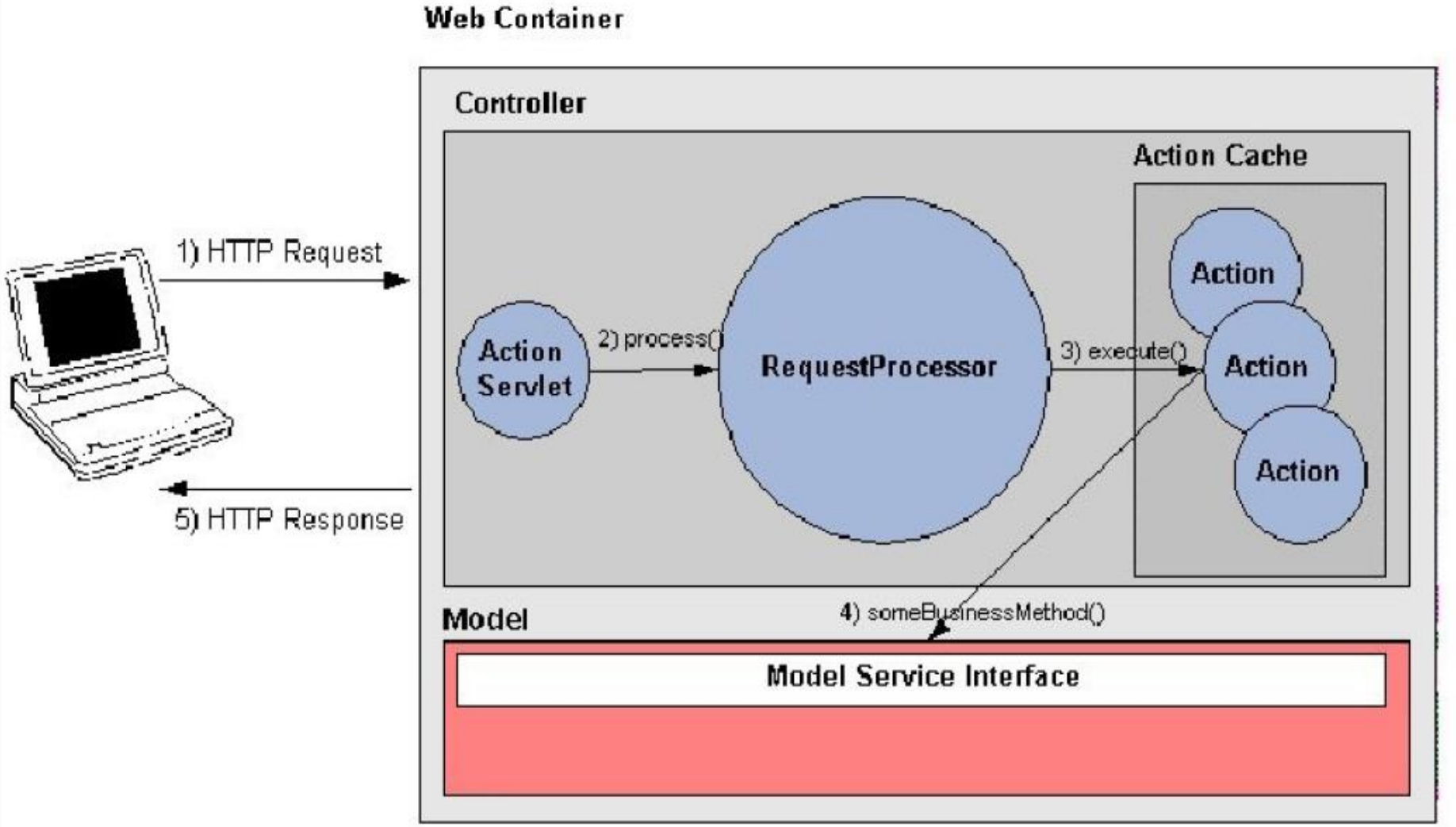
```
package submit;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.*;
public final class SubmitForm extends ActionForm {
    private String firstName = " ";
    private String lastName = " ";
    public String getLastName() {
        return this.lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

struts-config.xml: <form-beans>

```
<form-beans>
    <form-bean  name="submitForm“
                type="submit.SubmitForm"/>
</form-beans>
```

Action

Action Class Diagram



Action Class

- **Focus on control flow**
  - Process client request by calling other objects (BusinessLogic beans) inside its execute() method
  - Returns an ActionForward object that identifies a destination resource to which control should be forwarded to
  - The destination resource could be
    - JSP
    - Tile definition
    - Velocity template
    - Another Action

What is Action Class?

- **Java class that does the "work" of your application**
  - Handle request
  - Perform business logic

- **Can be simple or sophisticated**
  - Simple action class does handle business logic by itself
  - Sophisticated ones delegate business logic to Model components
    - Action class functions as a Facade pattern in this case

Example Action: Logon

- **Application needs to**
  - Authenticate a User
  - Establish a User Session
  - Error handling
- **Develop a “LogonAction”**

Developer Responsibility: Action Class

- extends **org.jakarta.struts.action.Action**

- overrides
  - **execute()** method (in Struts 1.1)
  - **perform()** method (in Struts 1.0)

execute(..) method of Action class

- Invoked by controller

```
public ActionForward execute( ActionMapping mapping,
                              ActionForm form,
                              HttpServletRequest request,
                              HttpServletResponse response)
                       throws Exception;
```

execute() method of Action class

- Perform the processing required to deal with this request
- Update the server-side objects (Scope variables) that will be used to create the next page of the user interface
- Return an appropriate **ActionForward** object

Example: Action Class

```
package submit;

import javax.servlet.http.*;
import org.apache.struts.action.*;

public final class SubmitAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request,  HttpServletResponse response) {
        SubmitForm f = (SubmitForm) form;
        String firstName = f.getFirstName();
        String lastName = f.getLastName();
        request.setAttribute(“firstName", firstName.toUpperCase());
        request.setAttribute("lastName", lastName.toUpperCase());
        return (mapping.findForward("success"));
    }
}
```

Design Guidelines of Action Class

- The controller Servlet creates only one instance of your Action class, and uses it for all requests
  - Action class has to be in multi-threaded safe
  - Use local variables (as opposed to instanace variables)

- Make Action class a thin layer
  - Use Model components for complex business logic handling

Example 2: Action Class

```
package submit;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public final class SubmitAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request,  HttpServletResponse response) {
        SubmitForm f = (SubmitForm) form;
        String firstName = f.getFirstName();
        String lastName = f.getLastName();
        if(firstName.equals(“Nwe”) && lastName.equals(“Ni”)){
            request.setAttribute(“firstName", firstName.toUpperCase());
            request.setAttribute("lastName", lastName.toUpperCase());
            return (mapping.findForward("success"));
        }else
            return (mapping.findForward(“failure"));
    } }
```

struts-config.xml: ActionMapping

```
<action-mappings>
    <action       path="/submit"
                  type="submit.SubmitAction“
                  name="submitForm“
                  input="/submit.jsp“
                  scope="request“
                  validate="true">
        <forward name="success" path="/submitSuccess.jsp"/>
        <forward name="failure" path="/submitFailure.jsp"/>
    </action>
</action-mappings>
```

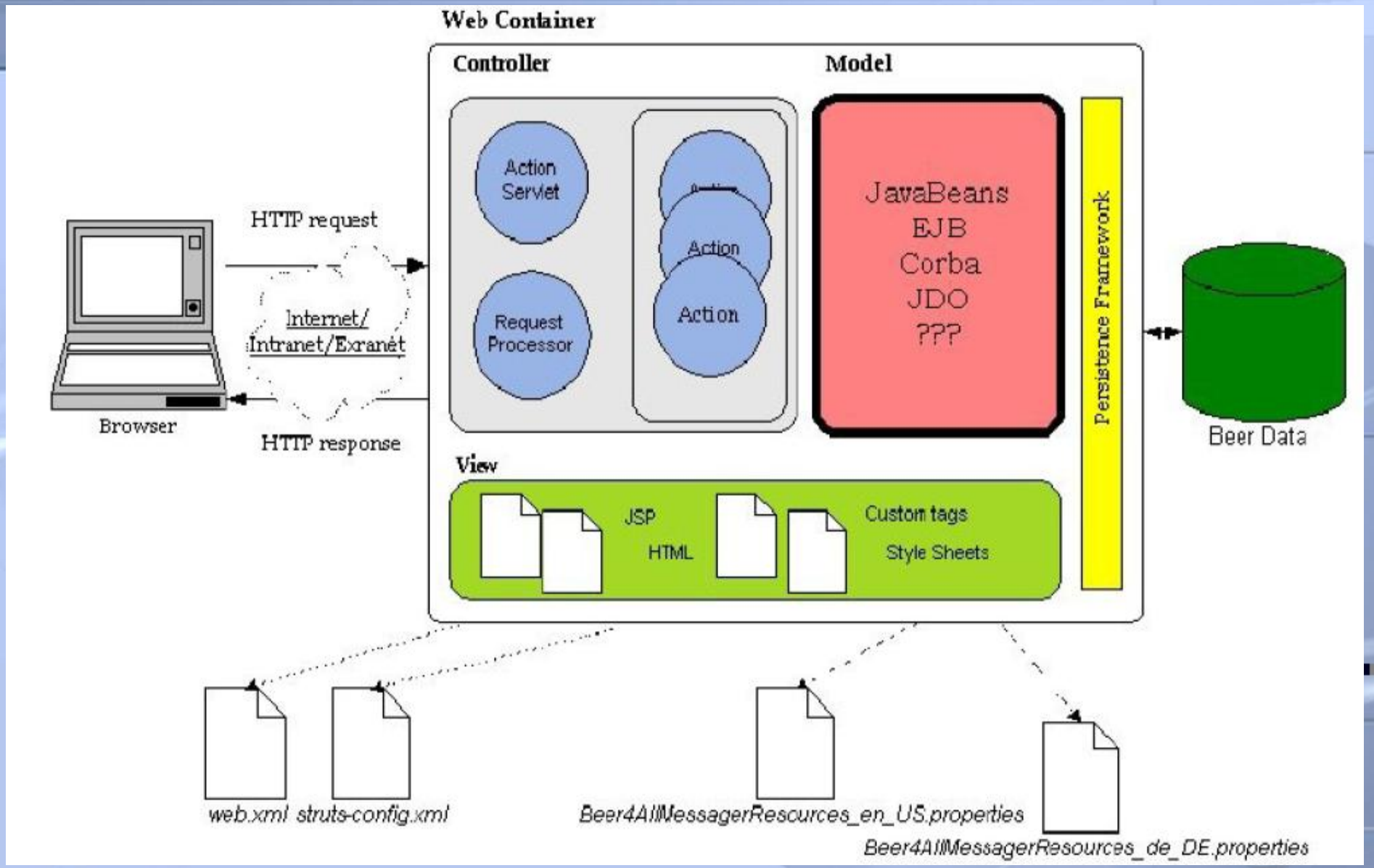
Pre-built Action Classes

- **ForwardAction**
- **DispatchAction**
- **IncludeAction**
- **SwitchAction**

Model Component

Model Components

- Model divided into concepts
  - Internal state of the system
  - Business logic that can change that state
- Internal state of system represented by
  - JavaBeans
  - Enterprise JavaBeans
  - POJO's
  - JDO
  - JDBC
  - Whatever


Web Container
Controller
Model
Action Servlet
Request Processor
Action
Action
JavaBeans
EJB
Corba
JDO
???
Persistence Framework
Beer Data
HTTP request
Internet/
Intranet/Exranet
Browser
HTTP response
View
JSP
HTML
Custom tags
Style Sheets
web.xml
struts-config.xml
Beer4AllMessagerResources_en_US.properties
Beer4AllMessagerResources_de_DE.properties

Model Components

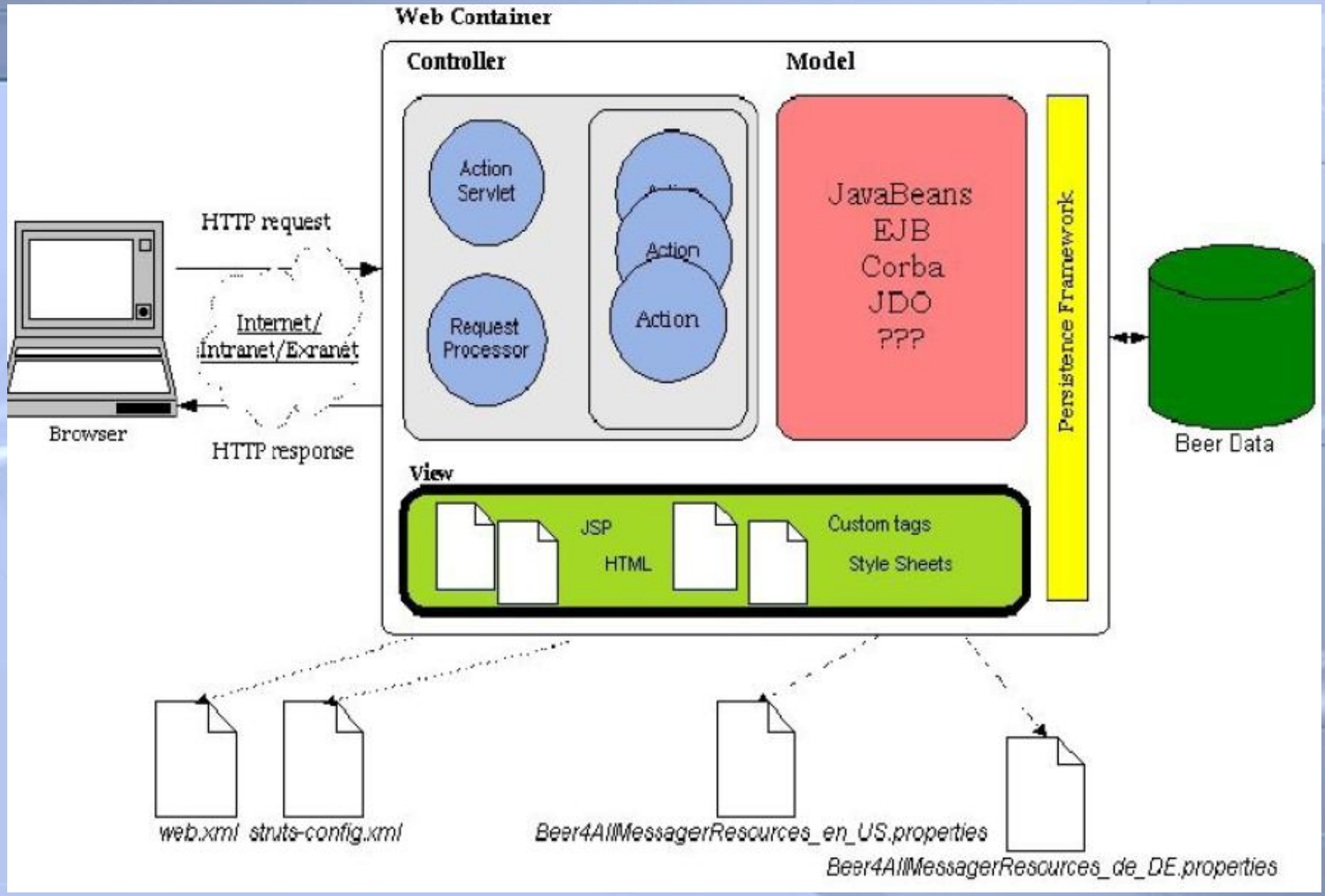
- **JavaBeans and Scope**
  - **Page** – visible within a single JSP page, for the lifetime of the current request.
  - **Request** – visible within a single JSP page, as well as to any page or servlet that is included in this page, or forwarded to by this page.
  - **Session** – visible to all JSP pages and servlets that participate in a particular user session, across one or more requests.
  - **Application** - visible to all JSP pages and servlets that are part of a web application

Model Components

- JSP pages and servlets in the same web application share the same sets of bean collections.

- Example
  - **Servlet code**
    - MyCart mycart = new MyCart(...);
    - request.setAttribute("cart", mycart);
  - **JSP page**
    - <jsp:useBean id="cart" scope="request“ class="com.mycompany.MyApp.MyCart"/>

View Component


Web Container
Controller
Model
Action Servlet
Request Processor
Action
Action
Action
JavaBeans
EJB
Corba
JDO
???
Persistence Framework
HTTP request
Internet/
Intranet/Extranet
Browser
HTTP response
Beer Data
View
JSP
HTML
Custom tags
Style Sheets
web.xml
struts-config.xml
Beer4AllMessagerResources_en_US.properties
Beer4AllMessagerResources_de_DE.properties

View Components

- **JSP** files which you write for your specific application
- Set of JSP **custom tag libraries**
- Resource files for **internationalization**
- Allows for fast creation of forms for an application
- Works in concert with the controller Servlet

View

- **ActionForward** object tells Servlet controller which JSP page is to be dispatched to.
- JSP pages use **ActionForm** beans to get output Model data to display
- Struts contains a series of **tag libraries**
  - Facilitates communication between HTML designers and developers
  - Facilitates dynamic Web content

Forms and FormBean Interactions

- If the user makes an error, the application should allow them to fix just what needs to be changed.

- With just JSP, you have to do

```
<input      type="text"
            name="username"
            value="<%= loginBean.getUsername() >"/>
```

- With Struts, you can do

```
<html:text property="username"/>
```

Example submit.jsp

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<html>
 <head><title>Submit example</title></head>
 <body>
      <h3>Example Submit Page</h3>
      <html:errors/>
     <html:form action="submit.do">
            First Name: <html:text property=“firstName"/><br>
            Last Name: <html:text property="lastName"/><br>
            <html:submit/>
       </html:form>
</body>
</html>
```

web.xml

Web App Deployment Descriptor (web.xml)

- Struts application is a Web application
  - Follows the same rule
  - Has to have web.xml deployment descriptor

- **web.xml** includes:
  - Configure **ActionServlet** instance and mapping
  - Resource file as <init-param>
  - servlet-config.xml file as <init-param>
  - Define the Struts tag libraries

- web.xml is stored in WEB-INF/web.xml

Example: web.xml

```
<web-app>
    <display-name>Advanced J2EE Programming Class Sample App</display-name>
     <servlet>
          <servlet-name>action</servlet-name>
          <servlet-class>
               org.apache.struts.action.ActionServlet
          </servlet-class>
          <init-param>
               <param-name>application</param-name>
               <param-value>ApplicationResources</param-value>
          </init-param>
          <init-param>
               <param-name>config</param-name>
               <param-value>/WEB-INF/struts-config.xml</param-value>
          </init-param>
    </servlet>
```

```
<!-- Standard Action Servlet Mapping -->
<servlet-mapping>
      <servlet-name>action</servlet-name>
     <url-pattern>*.do</url-pattern>
</servlet-mapping>
 <!-- Struts Tag Library Descriptors -->
 <taglib>
      <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
     <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
<taglib>
     <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
     <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
<taglib>
     <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
     <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>  </web-app>
```

Internationalization (i18n)

Internationalization

- Extends basic approach of java.util.ResourceBundle
  - **org.apache.struts.util.MessageResources**
- Allows specification of dynamic locale key on a per user basis
- Limited to presentation, not input
- Configure resource bundles in **web.xml** file

i18n : Developer responsibilities

- Create resource file for a default language.
- Create resource files for each language you want to support.
- Define base name of the resource bundle in an initialization parameter.
- In JSP page
  - Use <html:errors/> to display locale specific error messages

Resource files

- **MyApplication.properties**
  - Contains the messages in the default language for your server
  - If your default language is English, you might have an entry like this:
    - **prompt.hello=Hello**
- **MyApplication_xx.properties**
  - Contains the same messages in the language whose ISO language code is "xx“
    - **prompt.hello=Bonjour**

Example: ApplicationResource.properties

```
1   errors.header=<h4>Validation Error(s)</h4><ul>
2  errors.footer=</ul><hr>
3
4  error.lastName=<li>Enter your last name
5  error.address=<li>Enter your address
6  error.sex=<li>Enter your sex
7   error.age=<li>Enter your age
```

Configure in struts-config.xml,

```
<message-resources parameter="/WEB-INF/MessageResources" />
```

Example: web.xml

```
<servlet>
<servlet-name>action</servlet-name>
<servlet-class>
    org.apache.struts.action.ActionServlet
</servlet-class>
<init-param>
    <param-name>application</param-name>
        <param-value>com.mycompany.mypackage.MyApplication
    </param-value>
</init-param>
</servlet>
```

Validation

Validation: Developer responsibilities

- Indicate you want input validation as attributes of <action> element under
- <action-mapping> in servlet-config.xml file
  - **validate=”true”**
- Specify the JSP page that needs to be displayed when validation fails.
  - **input=”/errorpage.jsp”**
- Override **validate()** method within ActionForm class
  - optional

validate() method

- Called by the controller servlet
  - after the bean properties have been populated
  - but before the corresponding action class's execute() method is invoked
- Optional
  - default method returns null
- Syntax

  public ActionErrors **validate**(ActionMapping mapping,
  HttpServletRequest request);

Example: validate() method

- In Login application
  - **Make sure both “username” and “password” are entered**
  - **Make sure “password” is more than 6 chracters**

validate() method

- After performing validation
  - if **no validation error**, return **null**
  - If **validation error**, return **ActionErrors**
    - Each ActionError contains error message key into the application's MessageResources bundle
    - The controller servlet stores ActionErrors array as a request attribute suitable for use by the **<html:errors>** tag
    - The controller then forwards control back to the input form (identified by the input property for this ActionMapping)

ActionError Class

- Mechanism used to return errors during input validation.
- Encapsulate errors
  - message key used for text lookup from resource file
- Supports parametric replacement
- ActionErrors is a collection of ActionError (ActionMessage)

struts-config.xml: Validation

```
<action-mappings>

    <action         path="/submit"
                    type="hansen.playground.SubmitAction“
                    name="submitForm“
                    input="/submit.jsp"
                    scope="request“
                    validate="true">
          <forward name="success" path="/submit.jsp"/>
          <forward name="failure" path="/submit.jsp"/>
    </action>

</action-mappings>
```

In ActionForm

```
public final class SubmitForm extends ActionForm {
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest
    request) {
    ……..
     // Check for mandatory data
     ActionErrors errors = new ActionErrors();
    if (firstName == null || firstName.trim().equals("")) {
            errors.add(“FirstName", new ActionError("error.firstName"));
    }
     if (lastName == null || lastName.equals("")) {
            errors.add("Last Name", new ActionError("error.lastName"));
    }
    return errors;
}
```

Example submit.jsp

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<html>
 <head><title>Submit example</title></head>
 <body>
      <h3>Example Submit Page</h3>
      <html:errors/>
     <html:form action="submit.do">
            First Name: <html:text property="firstName"/><br>
            Last Name: <html:text property="lastName"/><br>
            <html:submit/>
       </html:form>
</body>
</html>
```

Error Handling

Input validation vs.Business logic Validation

- Perform simple validations using the ActionForm **validate()** method
  - Even this is optional
- Handle the "business logic" validation in the Action class

Application Errors in Action Class

- Capture errors from System State bean or Business Logic bean
- Create **ActionErrors** object and return
- **ActionServlet** can take action if a certain exception is thrown
  - Can be global or Action-based
  - Can either be a simple forward, or passed to a custom error handler class
  - Through defaut ExceptionHandler

View Selection

View Selection: Developer responsibilities

- In **struts-config.xml** file,
  - Indicate “to be forwarded JSP page” (or Tiles definition, another action, Velocity template) for each outcome via <forward> child element of <action> element or <global-forwards>
- In **execute()** method of Action class,
  - Return ActionForward object which is associated with a particular logical outcome

struts-config.xml

```
<action-mappings>

    <action         path="/submit"
                    type="hansen.playground.SubmitAction“
                    name="submitForm“
                    input="/submit.jsp"
                    scope="request“
                    validate="true">
          <forward name="success" path="/submit.jsp"/>
          <forward name="failure" path="/submit.jsp"/>
    </action>

</action-mappings>
```

In Action Class

```
package submit;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public final class SubmitAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request,  HttpServletResponse response) {
        SubmitForm f = (SubmitForm) form;
        String firstName = f.getFirstName();
        String lastName = f.getLastName();
        if(firstName.equals(“Nwe”) && lastName.equals(“Ni”)){
            request.setAttribute(“firstName", firstName.toUpperCase());
            request.setAttribute("lastName", lastName.toUpperCase());
            return (mapping.findForward("success"));
        }else
            return (mapping.findForward(“failure"));
    } }
```

Thank you!