Struts
An open source framework for web applications

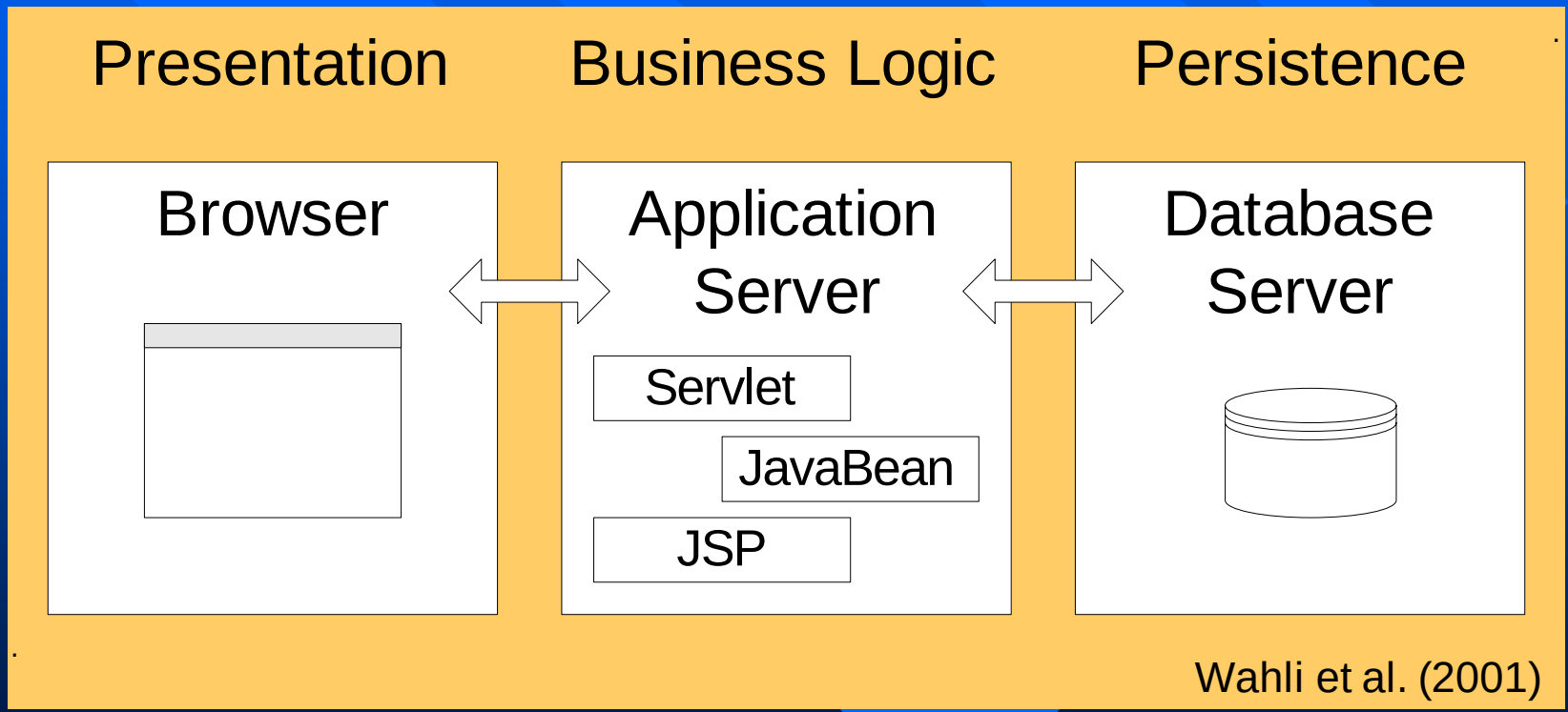
Jim Tyhurst, Ph.D.
Tyhurst Technology Group LLC

November 23, 2004

Overview

- N-Tier architecture
- MVC architecture
- Characteristics of web applications
- Struts framework
  - Architecture
  - Design practices
  - Coding idioms

3-Tier Architecture



Wahli et al. (2001)

N-Tier Architecture

Layer	Functionality	Role Stereotypes	Technique
Client	User Interface	Interfacer	HTML, JavaScript
Presentation	Page Layout	Interfacer	JSP
Control	Command	Coordinator	Servlet
Business Logic	Business Delegate	Controller	POJO
Data Access	Domain Model	Information Holder, Structurer	JavaBean
Resources	Database, Enterprise Services	Service Provider	RDBMS, Queues, ESB

N-Tier Architecture

Advantages

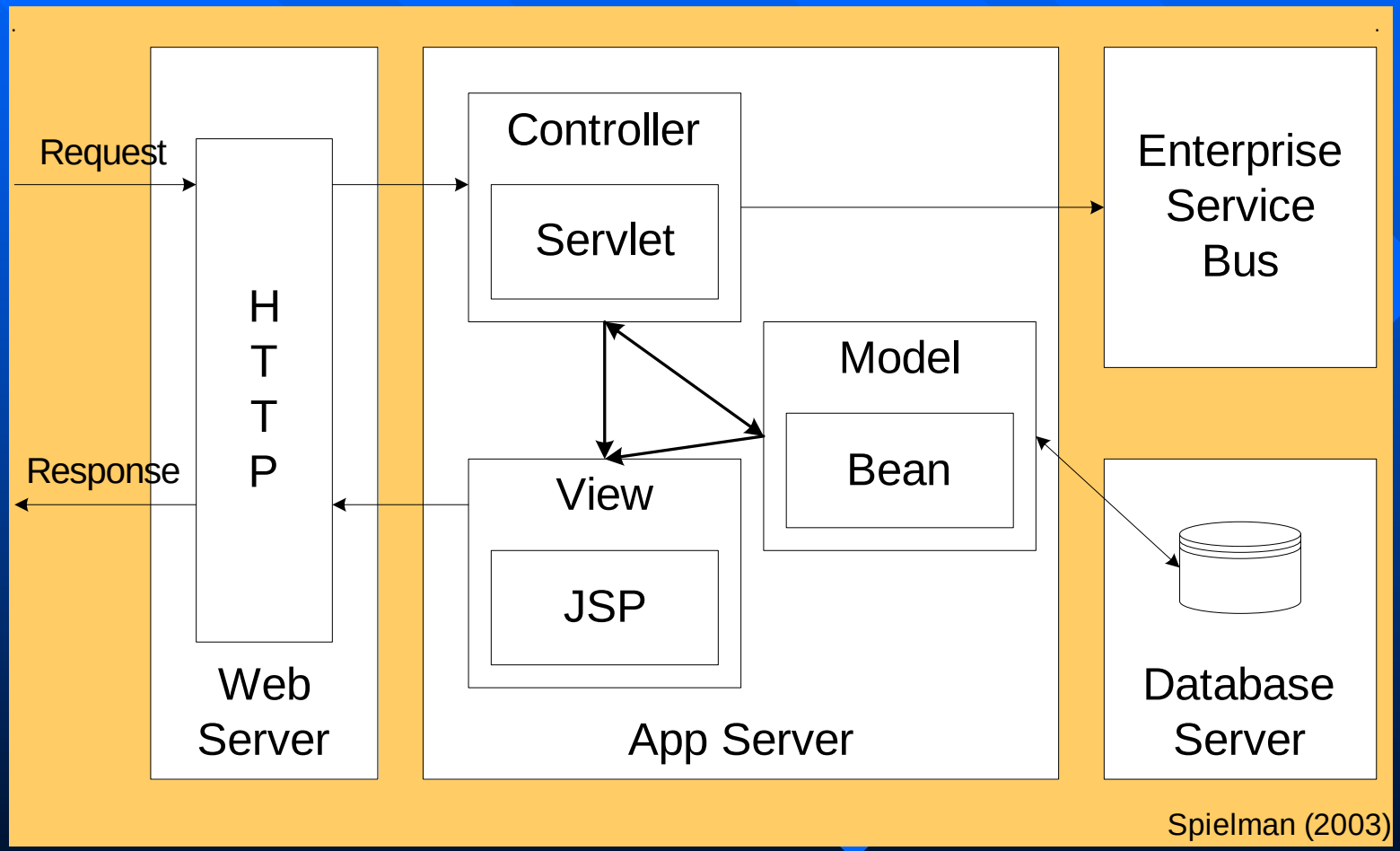
- Stateless connections enable scaling to large numbers of clients
- Enables reuse and flexibility by isolating responsibilities in independent layers
- Reduces complexity by limiting collaboration between adjacent layers

N-Tier Architecture

Disadvantages

- Limited client capabilities with HTML
- Increased response time to access business logic on the server
- Stateless connections result in additional server processing to reconstruct context for each request

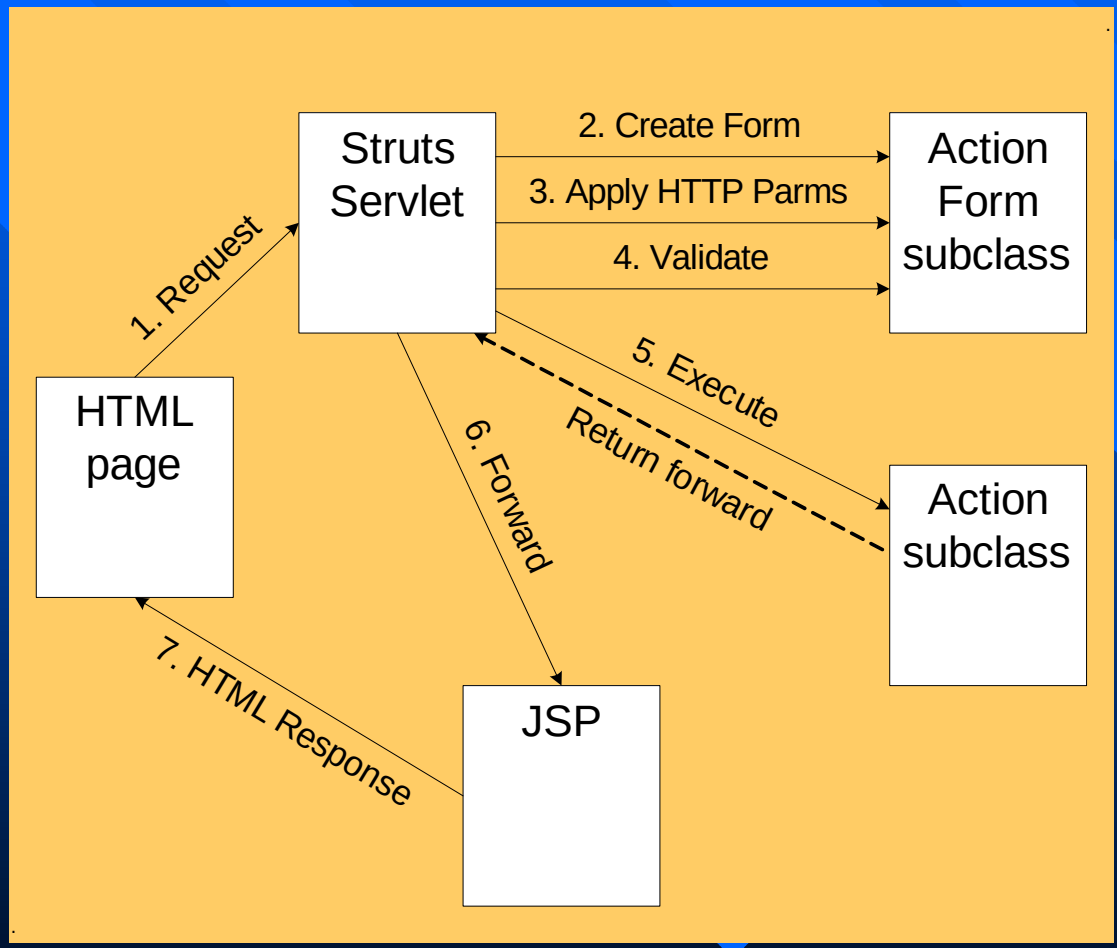
MVC Architecture



Characteristics of web apps

- Browsers display and capture text
- Domain model consists of objects
- Persistent store is usually a relational database
- Errors may occur on any of the layers
- Return original input for correction
- Page flow changes frequently
- Outline menus require complex navigation logic

Struts framework



Front Controller Servlet

- Controller manages common system services, e.g. security services, data validation
- Declarative navigation maps logical flow to implemented pages
- 'forward' options are specified in struts-config.xml, rather than in Java code

ActionServlet class

- Struts framework provides this controller
- Application can extend it if necessary, e.g. for shared authorization behavior

struts-config.xml

```
<action
   path="/PromptSubmit"
   type="com.tyhurst.sample.struts.action.PromptSubmitAction"
   name="promptForm"
   validate="true"
   scope="session"
   input="/page/prompt.jsp"
>
   <forward name="success" path="/GreetingDisplay.do" />
   <forward name="error" path="/ErrorDisplay.do" />
</action>
```

ActionForward Names

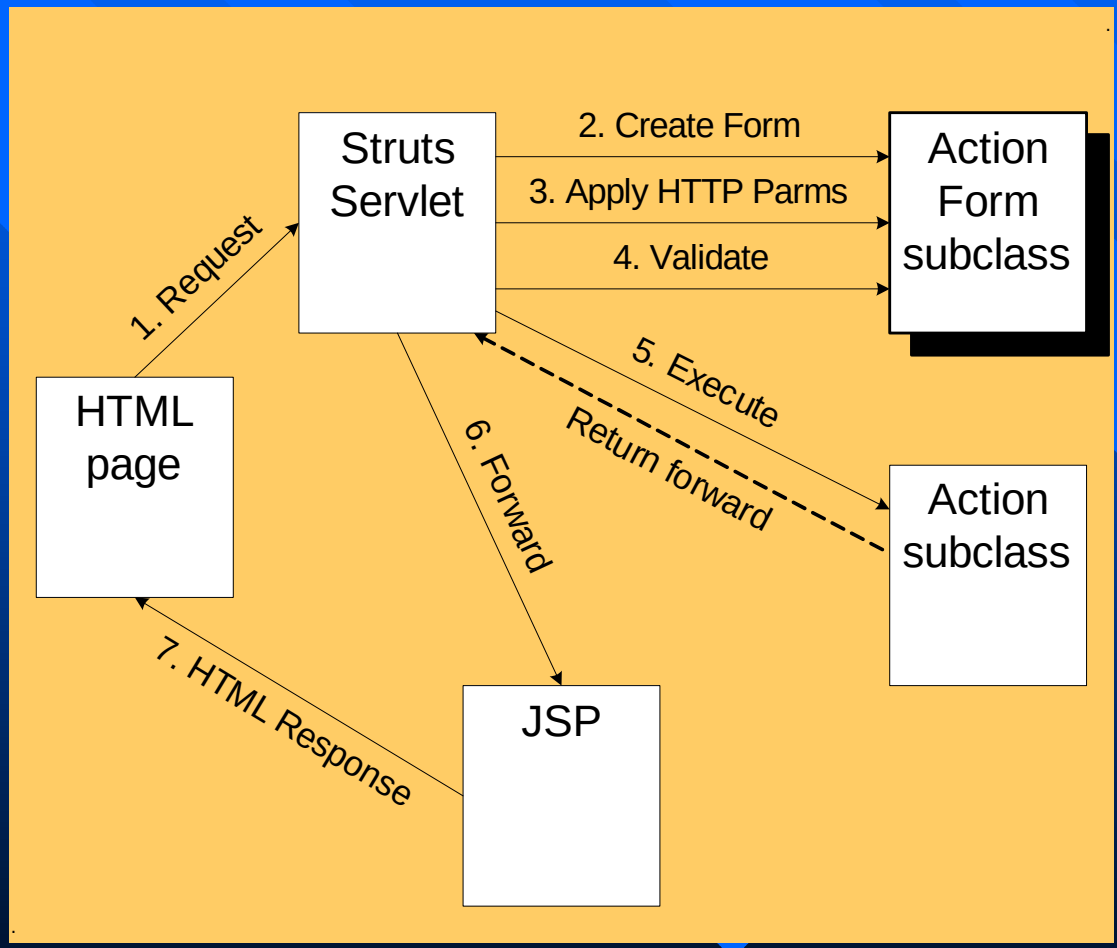
- Reference forward mappings with abstract logical terms
- Better: "success", "error", "nextPage"
- Worse: "loginSuccessful", "confirmPayment", "shippingAddress"
- Caveat: global forwards are usually to a specific page, e.g. "logout", “summarizeAccount", "payBills"

ActionForward References

Coding Idiom

- Use constants to refer to forward names
  - Config file:
    <forward name="error" path="/ErrorDisplay.do" />
  - Constant definition:
    public static final String ERROR = "error";
  - Action:
    forward =
    mapping.findForward(ForwardName.ERROR);

Struts framework - ActionForm



ActionForm class

- JavaBean to hold HttpServletRequest parameters
- Value Object for passing data from Action to JSP
- Subclasses override reset() and validate()

ActionForm

Design Practices

- Hold data as Strings, so no translation is needed from HttpServletRequest.
  - Mapping from/to domain model occurs elsewhere.

-OR-

- Hold object graph of domain objects.
  - Useful for quick implementation of read-only data, but JSPs become tightly coupled to domain model.

ActionForm

Coding Idioms

- Setter methods use normalize(String), so field values are always trimmed, non-null Strings

```
public void setCity(String city) {
    this.city = normalize(city);
}
```

ActionForm

… Coding Idioms

- Provide translation utilities, so JSP does not need to perform conversions.
  - Bean property:

```
public String getQuantity() {
        return quantity;
}
```

  - Helper method:

```
public int getQuantityAsInt() {
        return parseInt(getQuantity());
}
```

ActionForm

… Coding Idioms

- Override equals() method, so Action can compare incoming data to persistent data to determine if changes have occurred

```
if (!savedForm.equals(incomingForm)) {
    // Handle changes
}
```

ActionForm

Design Practices

- Create a super class to implement shared behavior

```
protected String normalize(String);
protected boolean isYear(String);
protected int parseStringAsInt(String);
```

ActionForm

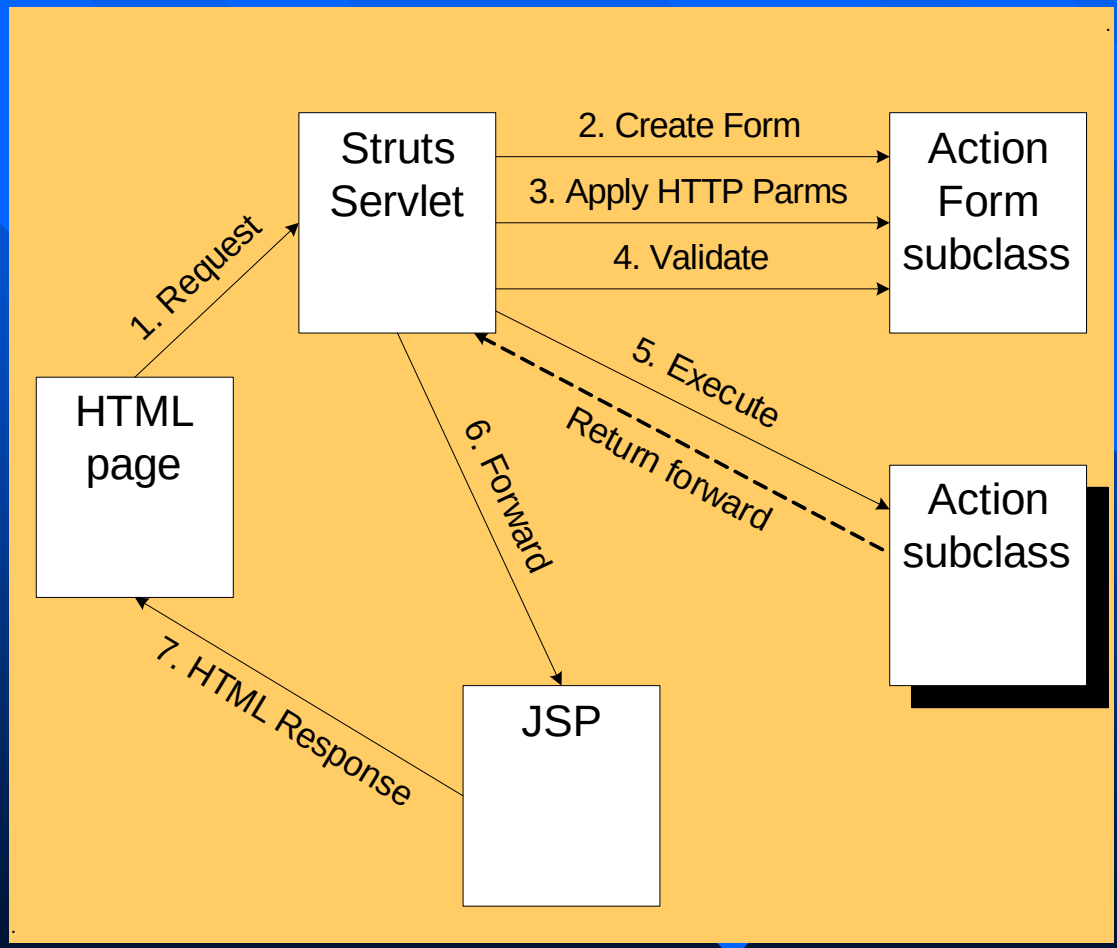
… Design Practices

- Use one Form for a single business “transaction”, even with multiple pages
  - For example, “wizard” that gathers data
- Use one Form per page when business logic or flow depends on evaluation of page-level data
  - For example, display progress in series of "wizard" pages.
  - Drawback: Value Object is tightly linked to presentation tier

One Form per page

Web Page	Corresponding Form
customerInfo.jsp	CustomerForm
subscriptionInfo.jsp	SubscriptionForm
paymentChoices.jsp	PaymentForm

Struts framework - Action



Action class

- Command pattern
- Request is an object
- Subclasses implement execute() method
- Controller Servlet delegates to consistent interface

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

Action

Design Practices

- One Action to display page;
  Another Action to handle submit;
  So, at least two Actions per web page

For example,

  – CustomerInfoDisplayAction
  – CustomerInfoSubmitAction
  – Maybe, DeleteCustomerAction

Action

… Design Practices

- Create application-specific super class to implement shared behavior
  - protected Logger getLog();
  - protected String getResponseErrorCode (Header header);
  - protected void addError (HttpServletRequest request, String errorKey, String errorResourceKey, Object[] replacementValues)

Action

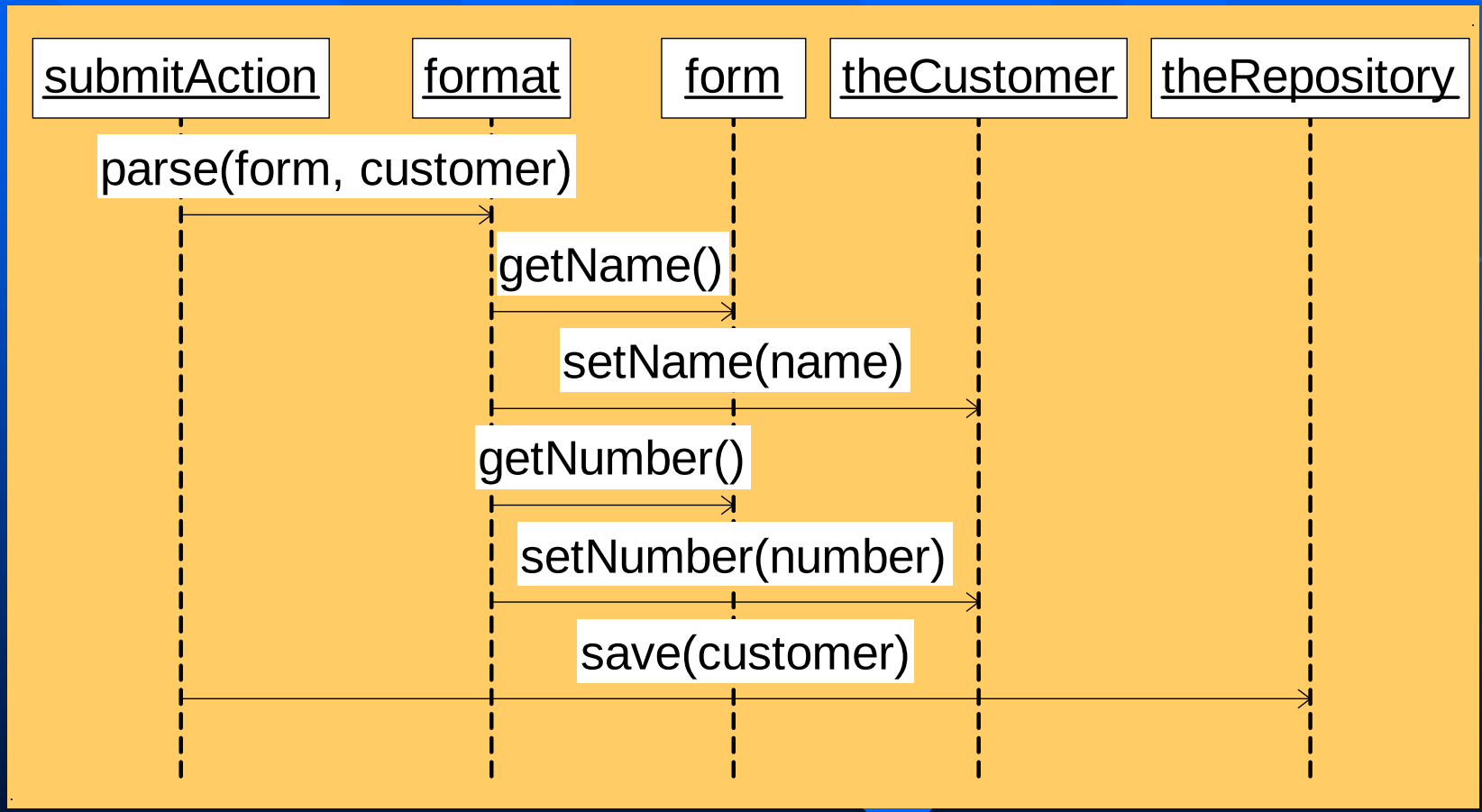
… Design Practices

- Use Format classes to map between ActionForm and domain model objects
  - Separation of responsibilities
  - Isolate and centralize mapping knowledge

Format class

- Format methods
  - public void format(
    Customer customer,
    SubscriberInfoForm form);
- Parse methods
  - public void parse(
    SubscriberInfoForm form,
    Customer customer);

Save user input with a Format class



Format class

- Centralizes mapping between ActionForm and domain model
- Makes maintenance easier
- Facilitates reuse
- Follows Java pattern for transformations, e.g. NumberFormat, DateFormat

Format class

Coding Idiom

- Write small methods, one per domain class
  - public void parse(
    SubscriberInfoForm form,
    Customer customer);
  - protected void parseMailingAddress(
    SubscriberInfoForm form,
    Address address)

Format class

Design Practices

- Use a super class to implement shared utility methods, e.g.
  - protected Date parseDate(String dateAsString);
  - protected String formatDate(Date date);

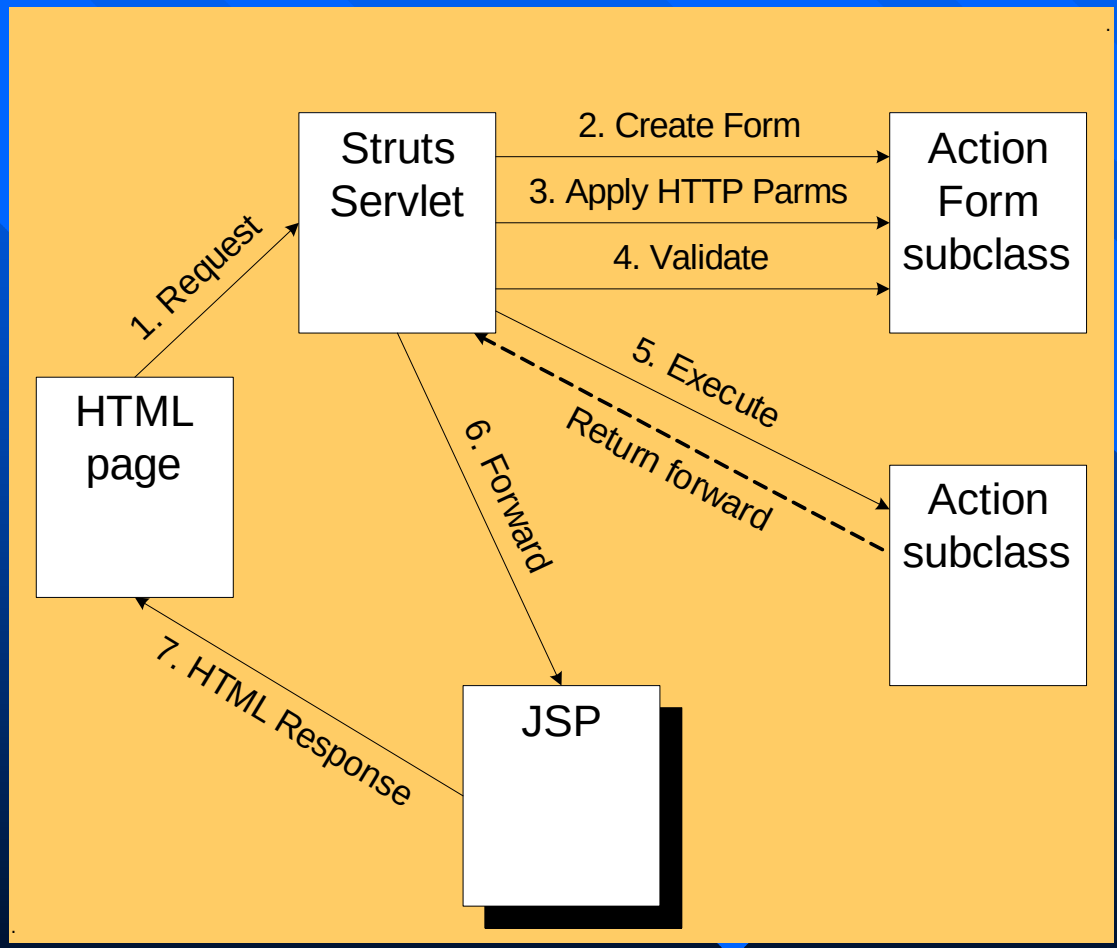
Domain Model

Coding Idiom

- Setters use nullify() methods.
  Set database fields null, rather than empty String or "--" undefined list value received from HTTP form

```
public void setCity(String city) {
        this.city = nullify(city);
}
```

Struts framework - JSP



JSP

- Focus on page layout, rather than formatting of individual fields
- So, let Format classes map from Objects to Strings
- Presentation only; no business logic
- But what about JavaScript for validation and conditional behavior? …

Validation at multiple levels

- Required fields
- Field syntax
  - Length
  - Type: alpha, numeric, date, part number
- Value range
- Dependencies between fields
  - Enable/disable input
  - Co-occurrence restrictions
- Validation in context of a transaction

Data Validation

Issues

- Division of responsibility
- Response time
- Security
- Reuse

Data Validation

Possible solution

- Syntax checking in ActionForm validate()
  - Override validate(); or
  - Jakarta Commons Validator framework
- Business logic and domain constraints available to Actions

Displaying Errors

- Action
  - protected void saveErrors (HttpServletRequest request, ActionErrors errors);
  - Protected void addErrors (HttpServletRequest request, ActionErrors errors);
- JSP
  - <html:errors/>

Other Struts topics …

- JSP custom tags
  - HTML: render ActionForm properties in HTML forms
  - Logic: conditional generation of output
  - Bean: access JavaBeans and their properties
- Tiles
  - Templates separate content from layout

Conclusion

- Struts framework is based on MVC
- Page flow is configured independent of business logic
- Action subclasses control business logic
- JSPs focus on presentation
- Action Forms represent data as Strings
- Format objects encapsulate mapping between Model and Form

References

- Deepak Alur, John Crupi, Dan Malks. Core J2EE Patterns. 2001.
- John Carnell, Rob Harrop. Pro Jakarta Struts. 2nd Ed. 2004.
- Chuck Cavaness. Programming Jakarta Struts. 2nd Ed. 2004.
- Ted Husted, et al. Struts in Action. 2003.
- Bill Dudney, Jonathan Lehr. Jakarta Pitfalls. 2003.
- Sue Spielman. The Struts Framework. 2003.

Links

- Struts home page
  http://struts.apache.org
- O'Reilly - Learning Struts
  http://www.onjava.com/pub/a/onjava/2001/09/11/jsp_servlets.html
- O'Reilly - Learning Struts 1.1
  http://www.onjava.com/pub/a/onjava/pub/a/2002/11/06/struts1.html
- Java Boutique - Coding your second Jakarta Struts Application
  http://javaboutique.internet.com/tutorials/Struts2
- Husted - Struts tips
  http://www.husted.com/struts/tips