Objects versus the Web

Naci Dai & Alan Knight

nacidai@acm.org

knight@acm.org

Copyright © 2001-2002

All Rights Reserved.

Introduction

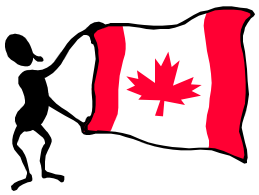
- **Part I**
  - Overview of technologies
  - What can go wrong
  - Separating Concerns
  - MVC for the Web
- **Break**
- **Part II**
  - Controllers and Actions
  - Multiple Input/Output Mechanisms
  - Validation
  - Flow Management

About the Presenters

- **Naci Dai**
  - *Working with objects (C++, Smalltalk, Java) since 1990*
  - *Object People/BEA 1995-2001*
  - *Teach Objects and Web Technologies*
  - *Java advocate against evil technologies*
  - *Founder of **ObjectLearn Group**, makers of fine and free J2EE development tools – www.objectlearn.com*
  - *Currently with **eteration** to show that objects are still good.*

About the Presenters

- **Alan Knight**
  - *Working with objects (Smalltalk, Java) since 1990*
  - *Object People/WebGain 1992-2000*
  - *TOPLink architect*
  - *Former representative EJB 2.0 expert group*
  - *Currently with Cincom Systems: Web technologies*
  - *"The Last Living Smalltalker" ☺*



Captain OOP Tackles the Web

- **Does the web need objects?**



- **If it does, what do they look like?**



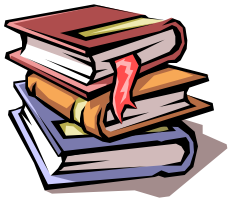
Objects vs. The Web

- **Object Virtues**
  - Well-factored code
  - Optimizing local issues (messages, GC)
  - Adaptability
- **Web Virtues**
  - Fast and simple to build
  - Works… Now!
  - Optimizing scalability
  - Adaptability (hmmm…..)

Goals

- **We will:**
  - Recall why we liked objects in the first place
  - Review web-based technologies, and the issues they raise
  - Do some “Architectural Cartography”
  - Examine various design choices, and see their trade-offs
  - See the building blocks of successful architectures
  - Be able to quickly assess frameworks from this perspective

Overview of Technologies



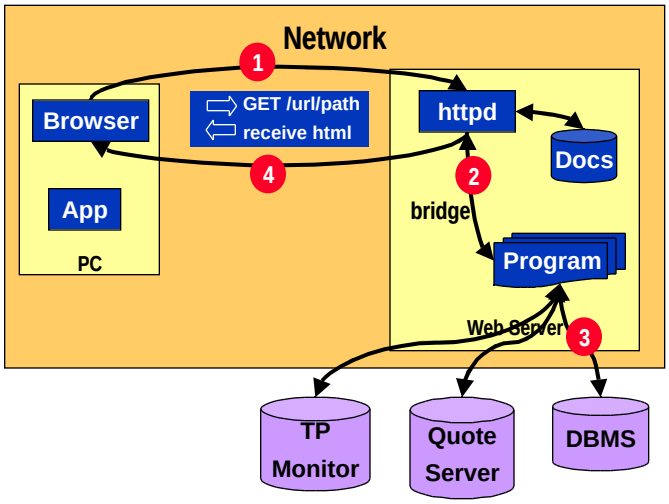
Objects Oversimplified

- **Everybody here knows what objects are**
  - Right?
- **Very neutral technology, but...**
- **Connotations**
  - Complex
  - Hard to Learn
  - GUI-focused
  - Large executables
  - Slow cycle times (build and deploy)
  - Old-fashioned?

The Web Oversimplified (1993 version)

- **It's about documents**
  - Static text
  - Search engines
- **Works well**
  - Very scalable
  - Lots of good, cheap servers
  - Simple model
  - For complex apps, embed client-side applications (ActiveX, Applets)
- **But very limited**
  - HTML is like a dumb terminal, only dumber!
  - Need dynamic content, interaction

The Dynamic Web



State vs. the Web

- **State is a problem**
  - Web requests are stateless
  - Difficult model for complex, interactive systems
- **A) Write your programs to be stateless**
  - not easy for most programs
- **B) Put state on the client (cookies, hidden fields, URL rewriting)**
  - network traffic
  - potential security issues
- **C) Just enough state to identify server state**
  - e.g. transient cookie with session key

Process Handling

- **Forking (i.e. CGI)**
  - Separate process per request
  - Expensive?
  - Complete isolation of request processing
- **One or few processes (e.g. Java, Smalltalk)**
  - One process can handle multiple requests
  - No forking overhead
  - Less isolation, crash can affect many requests
  - Better ability to share resources
- **Pre-forking (e.g. mod_perl)**
  - Single process per requests
  - Server maintains a pool of forked processes

What's "Program"

- **Lots of options**
  - Programs that generate/retrieve HTML
    - *CGI Scripts*
    - *Servlets*
  - Embedding scripts into HTML files
    - *ASP/JSP/PHP/Zope*
  - Others
    - *Special tags processed via framework and/or user code(JSP bean/custom tags, Enhydra XMLC, Apple WebObjects Templates, ColdFusion)*
    - *Automatic generation of web pages from other models (VisualWave)*

Representative Technologies

- **We will mostly talk about two general forms**
- **"Servlets"**
  - Simple programs generating HTML
- **"Server Pages"**
  - HTML pages with embedded code or special tags
  - Primarily ASP/JSP semantics
- **Smalltalk and Java implementations of these**
- **These are**
  - Well understood
  - Cover most of the issues
  - What we know...

Example Program (Java Servlet)

```
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest req,
                  HttpServletResponse res )
               throws ServletException, IOException {
       String name = req.getParameter("name");
       PrintWriter out = res.getWriter();
       res.setContentType("text/html");
       out.println("<HTML><BODY>");
       out.println("<h1>Hello "+name+"</h1>");
       out.println("</BODY></HTML>");
       }
}
```

Example Program (Smalltalk Servlet)

```
… #HelloServlet superclass: #{HttpServlet}...
doGet: request response: response
   | name out |
   name := request getParameter: 'name'.
   response contentType: 'text/html'.
   response writer
       nextPutAll: '<HTML><BODY>'; cr;
       nextPutAll: '<h1>Hello ', name ,'</h1>';cr;
       nextPutAll: '</BODY></HTML>';cr.
```

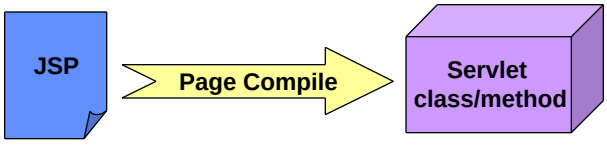

Example Program (Perl Script)

```
Hello.pl
#!/usr/bin/perl
use CGI;
$cgi = new CGI;
print $cgi->header(),
      $cgi->start_html(""),
      $cgi->h1("Hello ", $cgi->param("name")),
      $cgi->end_html();
```

Hmmm. These keep getting shorter.

Server Pages

- **HTML page with embedded code or special tags, processed on the server prior to delivery to client**
- **Server processing replaces these with content which is sent to the client**
- **Mechanism for coding dynamic content into otherwise static pages**
  - Pages look like HTML/XML with some code
- **Pages transparently compiled into programs or interpreted**
  - e.g.



JSP (Smalltalk)

JSP

```
<HTML>
<BODY>
The date is <%= Date today %>
</BODY>
</HTML>
```

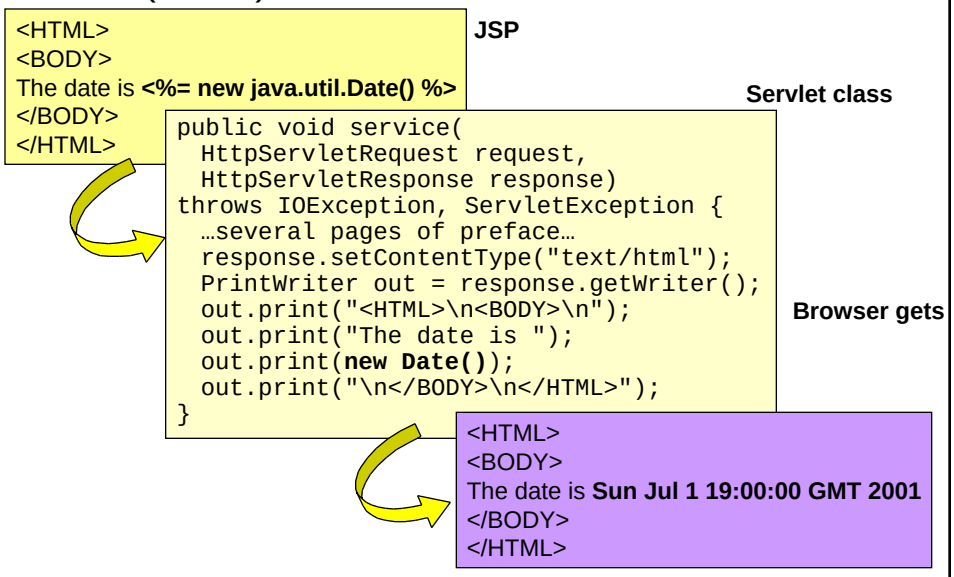
Servlet method

```
  response write: '<HTML>
<BODY>
The date is '.
  response write: ((Date today)
     displayStringForLocale: session locale)
  response write: (htmlChunks at: 2).
```

Browser gets

```
<HTML>
<BODY>
The date is Sun Jul 1 19:00:00 GMT 2001
</BODY>
</HTML>
```

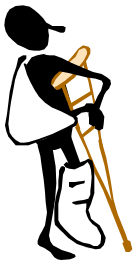
JSP (Java)



Resources

- **Databases**
- **Backend systems**
  - ERP/CRM
  - Existing “Legacy” systems
- **Components and services**
  - Things like EJBs, COM, JavaBeans, …
  - Back-end Web Services
- **Content-management systems**
- **Lots of issues but mostly not in our scope today**

What can go wrong?



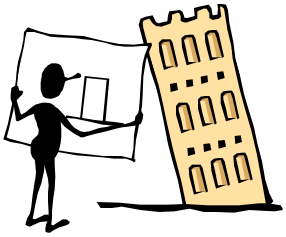
What goes wrong with software?

- It doesn't work



Could you be more specific?

- **Either**
  - It doesn't do what it's supposed to
  - It did, but now it should do something different and it's too hard to change
    - *(the interesting case)*



We can't make it work because...

- **We don't understand / it's too complex**
  - e.g. add a feature to the Quake engine
  - logic related to different features scattered through the code
  - grouping, separation of concerns
  - e.g. a name and address database tied to a GUI
  - e.g. an entire airline reservation system in seven server pages
- **Changing takes too much work**
  - e.g. change the look of a site with pages generated in code
  - duplication
  - word processing beginners
  - web design and programming culture gap
    - *throwaway/refactor, cut+paste/re-use*



Criteria

- **Separation of Concerns/Layering**
  - Unrelated functionality is not close together.
  - Changes to one area should not affect others.
- **Grouping**
  - Related functionality is close together.
  - Can be separate classes/methods, but not mixed with other functionality.
- **Duplication**
  - Once and only once
  - Changes should affect few other places, die down quickly.
  - Encapsulation: minimize access to representation
  - Structure: minimize dependence on class/method details

Separation of Concerns

- **Typical separate concerns**
  - Presentation
  - Application Logic
  - Domain Logic
  - Database
- **Other possibilities**
  - Distribution
  - Security
  - Transactions
  - Data Validation

Applied to the web

- **Scripts/Servlets group three separate concerns**
  - Get input
  - Apply application logic
  - Produce output
- **Also have problems in all three areas**
  - Input: duplication, unencapsulated
  - Application Logic: global data, duplication, number of scripts
  - Output: No separation of roles

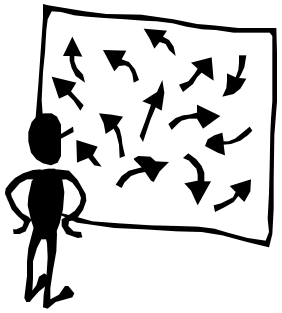


Servlet Issues - Getting Input

- **HTTP Information**
  - Lots of duplicate code dealing with request
  - All parameters are strings - lots of similar conversion code
    Parameter values entirely un-validated
  - Case sensitivity
- **Author has full access to request**
  - Knowledge of HTTP specific details
  - Programming at the level of the protocol
  - Coding and naming inconsistencies can develop
  - HTTP processing spreads into many layers of the application
  - MuhITTin syndrome

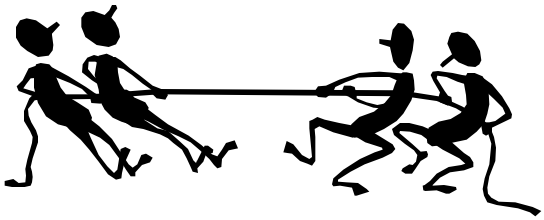
Servlet Issues - Application Logic

- **Where's the data**
  - Global data manipulated by many servlets
    - *Session*
    - *Database*
- **Where are the objects?**
  - Encapsulation?
- **Lots of similar or identical code**
  - Duplication
  - Overwhelming numbers of servlets/scripts
- **Control Flow**
  - Lose control of the order of events, even more so than in a GUI

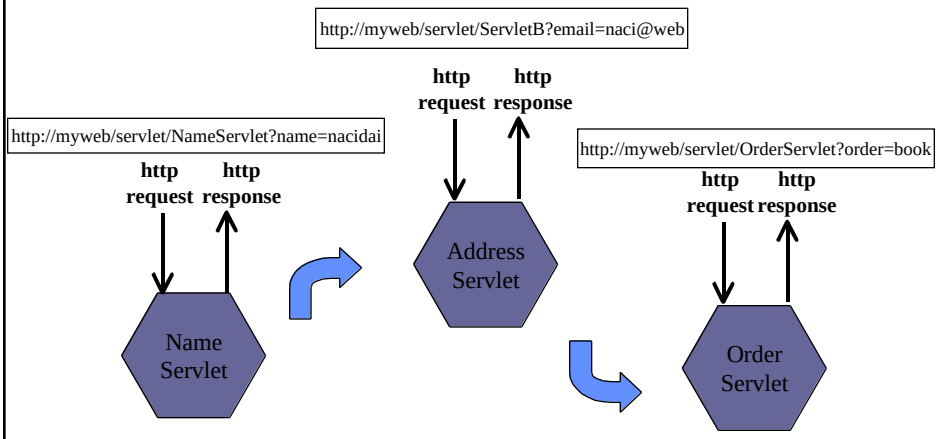


Servlet Issues - Produce Output

- **Want Separation of Roles**
  - HTML designers make nice pages
  - Programmers write nice programs
  - Serious cultural differences between them
- **HTML encoded into code**
  - Programmers must design or transcribe pages
  - HTML very difficult to change



Example: Naive Multi-Servlet Application



Example (continued)

- **What are the problems here?**

```
public class NameServlet extends HttpServlet {
public void doGet(... req, ... res )… {
    String name = req.getParameter("name");
    storeName(req.getSession(), name);
    PrintWriter out = res.getWriter();
    res.setContentType("text/html");
    out.println("<HTML><BODY>");
    out.println("<form action=/servlet/AddressServlet>");
    out.println("Email: <input type=text name=email>");
    out.println("<input type=submit name=Submit></form>");
    out.println("</BODY></HTML>");
}
```

Http Processing

Application Logic

Presentation

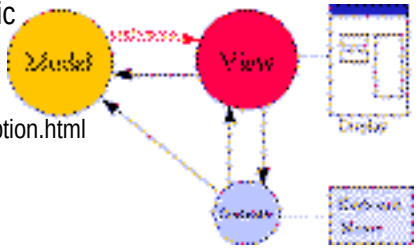
Example: Servlets are Nearly Identical

```
public class NameAddressServlet extends HttpServlet {
public void doGet(... req, ... res )… {
    String email = req.getParameter("email");
    storeEmail(req.getSession(), email);
    PrintWriter out = res.getWriter();
    res.setContentType("text/html");
    out.println("<HTML><BODY>");
    out.println("<form action=/servlet/AddressOrderServlet>");
    out.println("Email: <input type=text name=email>");
    out.println("<input type=submit name=Submit></form>");
    out.println("</BODY></HTML>");
}
```


Separation: Model-View-Controller (MVC)

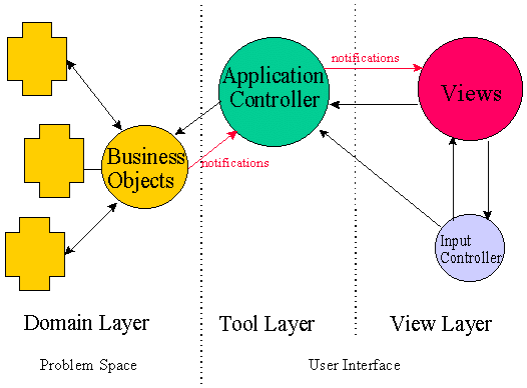


- **Old and Well-Established Architectural Pattern**
  - Frequently over-applied (now with 7 layers!)
- **Many different interpretations**
  - Not all are good ideas, or compatible
- **Historically used for early Smalltalk GUI's**
  - Model - application and domain logic
  - View - displays on screen
  - Controller - handles input
  - http://citeseer.nj.nec.com/krasner88description.html



Revised MVC (still GUI)

- **Change the meanings**
  - Hooking views directly to domain models didn't work well
  - Controllers processing input didn't make sense with native widgets
  - The term MVC had good connotations
- **So… re-arrange**



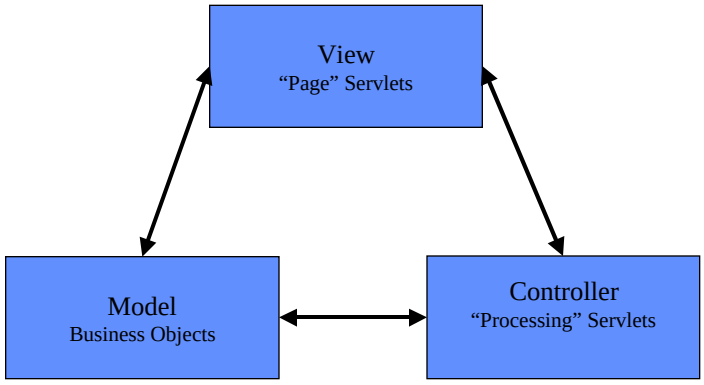
MVC for the Web?

- **Can we apply this pattern to web applications?**
- **What are the essential features**
  - Separation of presentation, control, and business logic
  - Allow each to be changed independently
  - "Models" suitable for multiple different uses
  - Sufficiently vague about its terms
- **Note: MVC normally implies the Observer pattern, which isn't as much use in this context.**

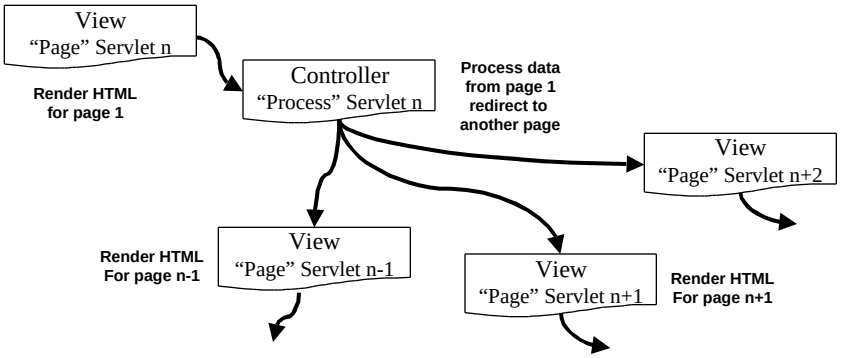
MVC with Servlets (v1.0)

- **We want to**
  - Split up the multiple functions of a script/servlet
  - Change business rules, presentation, and application logic independently
  - Support presentation for different clients (HTML, WAP, PDA, ???)
- **MVC paradigm fits well**
  - Model
    - *Domain objects implementing general business rules.*
  - View
    - *HTML/XML or WML*
  - Controller
    - *The "glue" layer. Application logic and flow.*
- **Note: Servlets only, no server pages yet.**

MVC v1.0 - Page & Process Servlets



MVC – Page & Process Servlets...



...MVC - Page & Process Servlets

```
PageServlet methods
doGet: request response: response
    | name |
    name := session getAttribute: 'name'.
    response write: '<HTML><BODY>'.
    response write: '<TITLE>Hello ', name.
    …and so on…
```

```
ProcessServlet methods
doGet: request response: response
    name := request getParameter: 'name'.
    session at: 'name' put: name.
    …do some processing…
    response redirectTo: 'servlet/NextPageServlet'.
```

Aside: Inter-Page Communication

- **Two basic mechanisms for changing from one page to another**
- **Redirect**
  - Client-side: send back a special command to switch URLs
  - New request, information must be preserved (session, url, etc)
- **Forward or Include**
  - Within the server, replace our output with another script, or include its output within our own
  - Gets more complicated with server pages (timing issues)
  - Server-side: Can attach "attributes" to the request.
    - *Attribute names are local to the request. Less conflict.*

Servlets with Request Attributes

```
PageServlet>>doGet: request response: response
    | name |
    name := request getAttribute: 'name'.
    response write: '<HTML><BODY>'.
    response write: '<TITLE>Hello ', name.
    …and so on…
```

request instead of session

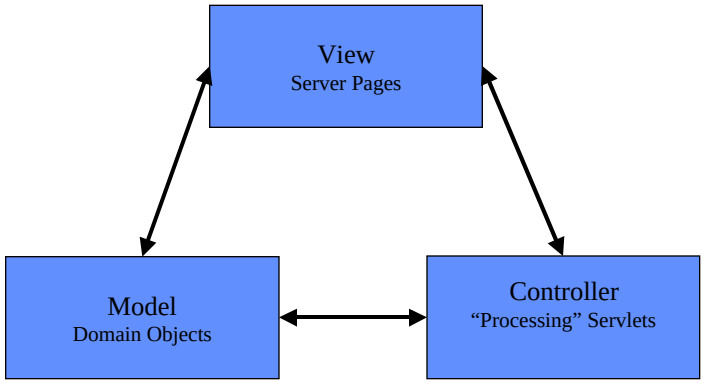
```
ProcessServlet>>doGet: request response: response
    | name dispatcher |
    name := request getParameter: 'name'.
    request setAttribute: 'name' to: name.
    (request getRequestDispatcher: 'servlet/Next')
        forward: request
        response: response.
```

forward instead of redirect

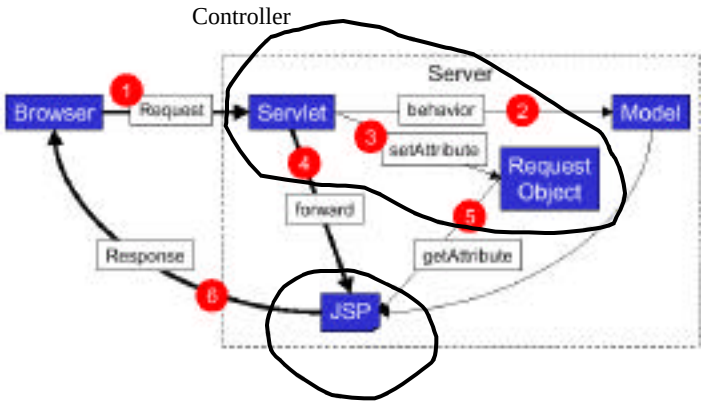
Issues with v1.0

- **Input processing separated from presentation**
- **But...**
  - Still generating HTML in code
  - Servlets do more than they should
  - Application logic is not separated from input
  - Hard to test in isolation
  - Still a lot of low-level programming and use of parameter names
- **Need to further refactor**
  - Minimize generation of HTML in code - server pages
  - Similar idea to GUI builders

MVC 1.1 Servlets and Server Pages



MVC – Server Pages



MVC – Server Pages...

```
public class ProcessingServlet extends HttpServlet {
public void doGet(... request, ... response )… {
...
request.          e(" account", account);
```

```
}
<%BankAccount
account=(BankAccount)request.getAttribute("account");%>
<HTML><BODY>
<H1>Account no: <%= account.getAccountNumber() %></H1>
<H1>Balance: <%= account.getBalance() %></H1>
...
</BODY></HTML>
```

showaccount.jsp

MVC – Server Pages

- **Good things**
  - Presentation is separated from application/domain logic
  - There is no HTML generation in code
    - *Well-structured server pages shouldn't need programmers <grin>*
    - *Danger of too much code in the HTML!*
  - Use objects to provide the content.
- **But…**
  - Application logic is still in servlets (no model/controller separation).
  - Low-level http access duplicated in many places
    - *inconsistencies*
    - *maintenance problems*
  - One servlet per user action = LOTS of servlets

Managing Server Pages

- **Server pages are good places to write HTML/XML/WML**
  - Complete control: can do anything that can be done with a web page
- **Server pages are not good places to write code**
  - Editors are not code-friendly
  - Lack of development environments and debugger support
  - Even with debuggers, generated code is hard to follow
  - Reduced readability
  - Code in two places - pages and regular code
    - *configuration and versioning issues*

It can be hard to eliminate all code from server pages

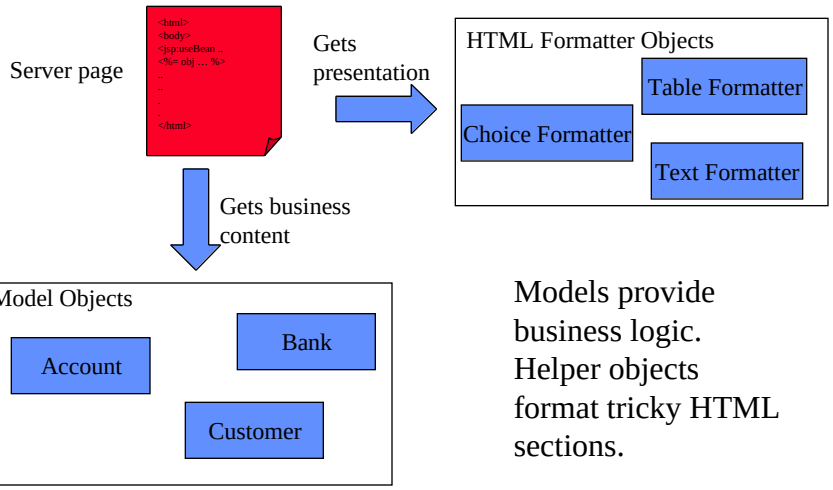
A JSP nightmare

```
<% String accttType = request.getAttribute("acct_type"); %>
<select name="Account Type" size=2>
<option value="Savings"
  <%= (accttType.equals("Savings")? "selected" : "") %>
>Savings</option>
<option value="Checking"
   <%= (acctType.equals("Checking")? "selected" : "") %>
>Checking</option>
</select>
<table>
<% for(int i=0; i< num; i++){ %>
<tr>…</tr>
<%}%>
```


Aside: Keeping Code Out Of Pages

- **Minimize the size of expressions**
  - `emp.formattedName()` rather than `emp.lastName()+ ", " + emp.firstName()`
  - Avoid loops and conditionals if at all possible
  - Consider use of formatter methods/objects
- **BUT…**
  - the tradeoff that you may end up turning a static HTML page into a program

Formatters



Models provide business logic. Helper objects format tricky HTML sections.

Formatters

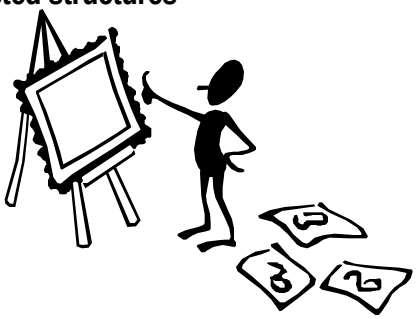
```
ChoiceFormatter>>showName: name choices: aCollection
      size: anInteger selected: anObject
    self writeSelectWithName: name size: anInteger.
    aCollection do: [:each |
        self writeOption: each selecting: anObject].
    self write: '</select>'.
    ^self contents.
writeOption: anObject selecting: selected
    outputStream
        nextPutAll: '<option value="';
        nextPutAll: anObject displayString, '" '.
    outputStream nextPutAll: (anObject = selected
        ifTrue: [' selected>'] ifFalse: ['>']).
    outputStream nextPutAll: '</option>'.
```

Can abstract details of HTML generation

Works for arbitrary objects

Formatters - Issues

- Good - Helps move code out of JSP
- Bad – Page is further removed from normal HTML. Less designer control over formatting
  - **Fine (probably) for <select>**
  - **Not so good for <table>**
  - **Bad for complex or nested structures**



More JSP

- **Special tags allow for more modular JSPs**
  - <jsp:include ..>, <jsp:forward >
- **Repetitive scripts can be encapsulated in "tags"**
  - Taglibs – Typically "8-20"?? tags for an application handle most cases
  - Less code in html/wml/xml
- **Can preserve static properties of the page**

Custom tag. Internal text is ignored at run-time

```
<foo:optionTag object="customerList" selection="currentCust">
  <select name="cust" size="3">
    <option value="One">John Doe</option>
    <option value="Two" selected>Jane Doe</option>
  </select>
</foo:optionTag>
```

Example HTML. Only part visible statically.

Codeless Server Pages

- **Other ways of separating presentation from code**
  - Enhydra XMLC
  - XSP - Apache XML Project Cocoon
- **Create pages via transformation**
  - Façade to DOM manipulation in Java
  - XSLT plus special tags and a SAX engine
- **Reduce or eliminate scripting in HTML**
- **May have better WYSIWYG editing (by using standard tags)**
- **More abstraction supported/required**

Alternatives: Rich Clients

- **We've assumed all meaningful processing on the server**
- **We can move intelligence to the client with programs or partial programs**
  - Java Applets/ActiveX Controls/Plugins
  - Serialize data/objects between server and client
- **Advantages**
  - Much richer interface on client side
- **Disadvantages**
  - Differences in clients may become important
  - More bandwidth, slower startup
  - More complex programming model (perhaps much more)
  - More care required on security issues
  - Less customer acceptance

Javascript Clients

- **An interesting alternative, but not so much used**
  - Pavel Simakov, Frogware
- **Web designers are used to writing Javascript code, even for moderately complex operations.**
  - So, put all your data into Javascript objects and let them display/validate/create those objects.
  - Put static example objects at the top of the page in a known format
  - Generate dynamic example objects at runtime to replace them

```
<HTML>…<Script>
var customers = new Array(5);
var currentCust = new Customer();
Customer.setName(“Fred”);
… </script>… <BODY> …
<script>language=“javascript”>document.write("<CENTER>"+customer[i]
.getName()+“</CENTER>");
</script>
```


JavaScript Clients

- **Advantages**
  - Keeps web pages entirely for the web designers.
  - Pages are statically viewable/editable
- **Disadvantages**
  - Javascript details vary between clients
  - Synchronization issues between client and server code definitions
  - Debugging/testing the client end is difficult
- **In production and working well**

Recap

- **Server Page Advantages**
  - Can be built primarily by non-programmers, with good control
  - Help separate code from presentation
  - Reasonably simple to use
- **Server Page Disadvantages**
  - Still hard to get complete separation
  - Development environments are weak!
    - *Hard to debug/manage/test*
- **There are alternatives**
  - But none yet with wide-spread acceptance and use.

Break



Controllers and Actions



Recall

- **Software is hard (even Web software)**

Good Things	Bad Things
Cohesion	Duplication
Separation of concerns	Dependency
Testability	

- **A simple servlet combines all the layers**
  - Input processing
  - Application flow
  - Domain logic
  - Presentation

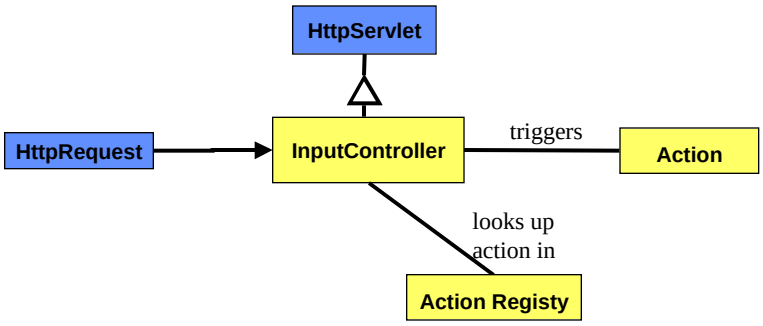
Recall (2)

- **We applied Model-View-Controller**
  - Use servlet for input parsing/application flow
  - Redirect/forward to server page for presentation
    - *Keep server page logic simple by delegating operations, custom tags, etc.*
  - Business objects as “model”, independent of input/presentation
- **But we also want to**
  - Avoid inflationary growth of servlets
  - Avoid direct http programming
  - Further separate concerns
  - Be able to maintain and re-use pieces in independently

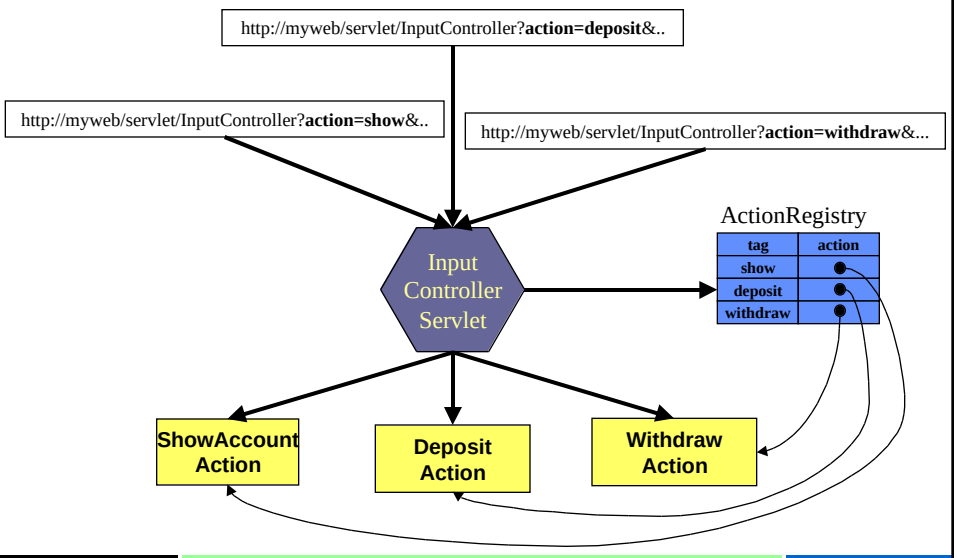
Motivations

- **We still have application logic in our servlets. This is a problem.**
  - Application logic coupled to HTTP
  - Hard to test in isolation
  - Not well structured
    - *One per user action rather than following the domain*
    - *Easy to have duplicate code*
    - *Hard to follow when there are too many*
- **Recall MVC: Controllers handled input and delegated actions**
  - Create controller servlets that only delegate
  - Since they delegate, one servlet can handle many tasks
  - Translate state into a more generic form than HTTP

Controller with Actions



Controller



Actions

Action represents the application logic for a single business event. ***ActionRegistry*** maps the action tag name stored in a request to an instance of an action. ***InputController*** handles requests from clients, maps them to an action and triggers the action.

- **InputController,**
  - HttpServlet that handles incoming requests
  - ActionRegistry
    - *Mapper that parses actions tags from http-requests and returns the corresponding action*
- **Action**
  - Defines the action to be taken

InputController...

```
public class InputController extends HttpServlet {
    public void service( HttpServletRequest req,
                         HttpServletResponse res )
        throws ServletException, IOException {
        //Acquire an action
        Action action = getRegistry().parse(req);
        //Dispatch the action and forward to
presentation layer
        String nextPage = action.trigger(req, res);
        this.forwardPage(nextPage);
    }
}
```

Implementing Actions v1.0

- **Each action is a class with one method**

```
public class ShowAccountAction implements Action {
public String trigger(HttpRequest req){
    String accountNo = req.getParameter("acct_num");
    Account account = readAccount(accountNo);
    req.setAttribute("account",account);
    return "/jsp/showaccount.jsp";
}
```

- **Instead of many servlets we have one servlet, many actions**
  - Better testability
  - Actions still access HTTP directly
  - Still a lot of one-method classes

ActionContext

- **Introduce an object to abstract away state access**
  - Hide getParameter calls and avoid session variables
  - Access via methods
    - *Remove need for naming conventions*
    - *Type-safe (if you like that kind of thing)*
  - Store application state to avoid get/setAttribute calls
  - Can potentially do validation
- **Store this object in the session**
  - Now we can remember previous parameters
    - *e.g. which account the previous action acted on*
  - Can cache computed state
  - Can better remember our history

ActionContext

```
account
    "Basic version. Get the account number from the request
and look up the appropriate account"
    ^self readAccount: self accountNumber.
```

```
accountNumber
    num := request getParameter: 'acct_no'.
    ^num isNil
        ifTrue: [lastAccountNumber]
        ifFalse: [num].
```

Use previous state for defaults

InputController v2.0

```
triggerAction: action
    context := self createContextFor: action.
    action trigger: context.
    self forwardPage.
```

```
forwardPage
"Forward to the url (normally a server page) that the
action stored in the context"
    | url dispatcher|
    url := context viewPage.
    dispatcher := self getRequestDispatcher: url.
    dispatcher forward: request response: response.
```

Action decides the next page

Creating an ActionContext

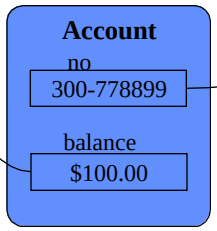
Applications/actions should be able to have different contexts

```
InputController>>createContextFor: anAction
    context := anAction contextClass new.
    context request: request.
    ^context.
```


Example Page

showaccount.jsp

```
<% BankContext context=(BankContext)session. getAttribute ("bankContext");
account = context.getAccount();%>
<HTML> <BODY>
<H1> Account no: <%= account. getAccountNumber() %></ H1>
<H1> Balance:      <%= account. getBalance() %></ H1>
...
</ BODY></ HTML>
```



Implications

- **Only one Servlet is needed**
  - Consistent implementation, limited exposure to http
- **HTML is in the views (server pages) not code**
- **Better testability, actions can be run directly**
- **But…**
  - Still lots of actions, we've just moved the servlet code into separate objects
  - Code still structured according to user actions
  - Additional complexity for simple applications

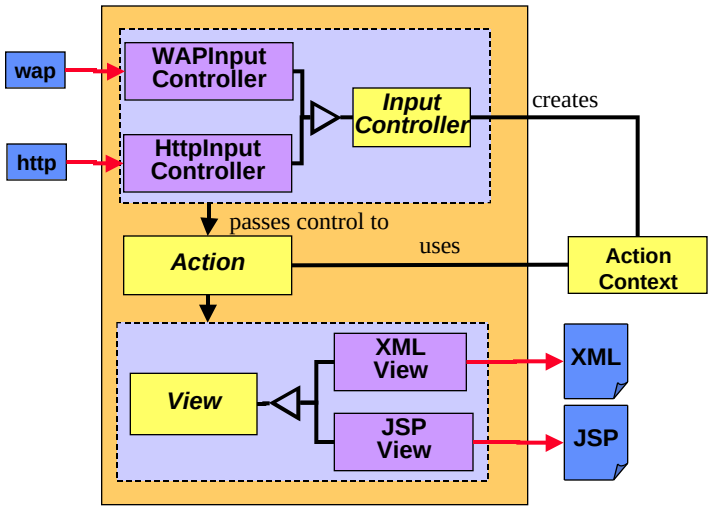
Views



Motivations

- **InputController is still doing several things**
  - Input processing
  - Action/Event detection
  - Output processing
  - All in one!
    - *Not suitable for multi-channel, multi-protocol, multi-access sites*
- **Split output processing for multiple layers**
  - Two levels of views
  - "View" = policy object that knows how to render for an output mechanism
  - "Page" = the output format itself, JSP, XML, etc.

Multiple Input/Output Mechanisms



Input Controller

- **Parses specific protocols to extract input data**
  - http, wap, tcp/ip, rmi, corba, web service protocols
- **Identifies the Action**
- **Builds the ActionContext**
- **Invokes the Action**

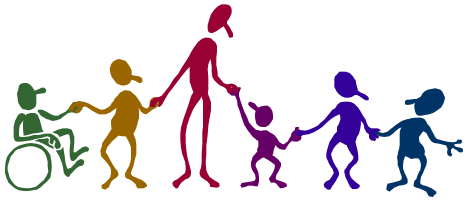
```
public void service( HttpServletRequest req,
        HttpServletResponse res )
    throws ServletException, IOException {
    String actionID = parseActionID(req);
    Action action = getController().getAction(actionID);
    ActionContext context = this.createContext(action,req);
    String nextPageID = controller.trigger(action,context);
    getView().forwardPage(context,nextPageID);
```


View

- **Knows the mechanisms for forwarding and preparing presentation in a specific format/technology**
- **Determines presentation format**
  - JSP, XML presentation, data sent to a GUI or web-service client
  - May handle sub-protocols and formatting
    - *Uses different JSPs to generate HTML or WML*
    - *Internationalization*

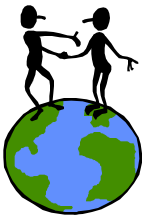
Aside: Multi Access Sites

- **Multiple Languages**
  - i18n (internationalization) support for messages
  - view object can forward to language specific locations
  - automatically detect the language
- **Multi Device**
  - WAP/WEB/Palm type devices have different requirements
  - view forwards to device specific locations



Language Detection

- **Automatically detect the language**
  - Request contains information about the default preferred language of the client
- **Be able to manually overwrite the preferred language**
  - Avoid the Internet-cafe problem



```
Locale locale = httpRequest.getLocale();
String language = locale.getLanguage();
```

Removing hard coded text

- Must remove language specific text from server pages or code
- Internationalized Message Bundles can be accessed from code

messages_en.properties

```
register.form.nameField=Name
register.form.lastNameField=Surname
```

messages_tr.properties

```
register.form.nameField=Ad
register.form.lastNameField=Soyad
```

- getI8NMessage returns the string for the current language

```
<% String label =
   context.getI8NMessage("register.form.nameField"); %>
<%=label%> <input type=text name="register.form.name" >
```

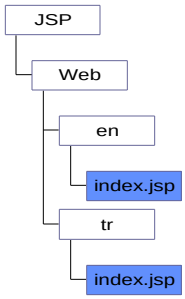

Multiple languages: Application Controller

```
ApplicationController  Class

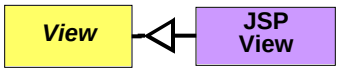
public String getI8NMessage(String id)
{
  return getI8NMessage(id, this.getLanguage());
}
public String getI8NMessage(String id, String language)
{
  PropertyBundle props = i8nMessages.get(language);
  return props.getString(id);
}
```

Multiple Languages: Extending View

- **Language is not all in text**
  - Layouts, graphics, content can be different
- **Different pages for different languages**
  - Standardize directory structures



View – Handling multiple languages




```
public void forward(ActionContext context,
    HttpServletRequest req,
    HttpServletResponseresp, String pageID )
    throws ServletException, IOException
{
    String devicePrefix = “/jsp/web/”;
    String langPrefix = context.getlanguage();
    String pageUri = this.getPage(url);
    RequestDispatcher dispatcher =
context.getDispatcher(devicePrefix + langPrefix + pageUri);
    dispatcher.forward(context.get, resp);
}
```

Multi Device - Detection



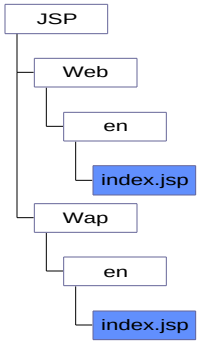
- **Automatically detect the device**
  - Request contains information about the browser type
    - *Each device typically inserts its own type*
  - WAP
    - *Request contains an attribute called "msisdn"*
- **Parse and store the device type into the context**

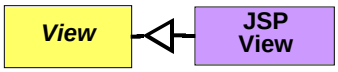
Multi Devices

- **Devices have significantly different requirements for specifying and rendering pages**
- **Some standards**
  - HTML – Web browsers and handheld PCs
    - *Same syntax but content must be different*
  - WML
    - *WAP enabled wireless devices, mobile phones*

View – Extending for Mobile devices

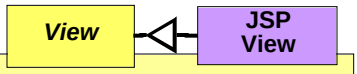
- **Different methods for handling mobile devices**
  - Dynamic XML translation with XSL for each device type
    - *Transcoding Pages*
  - Explicit





View

- **Adding device prefix**




```
public void forward(ActionContext context,
    HttpServletRequest req,
    HttpServletResponseresp, String pageID )
    throws ServletException, IOException
{
    String devicePrefix = context.getDevice();
    String langPrefix = context.getlanguage();
    String pageUri = this.getPage(url);
    RequestDispatcher dispatcher =
context.getDispatcher(devicePrefix + langPrefix + pageUri);
    dispatcher.forward(context.get, resp);
}
```

Application Controllers

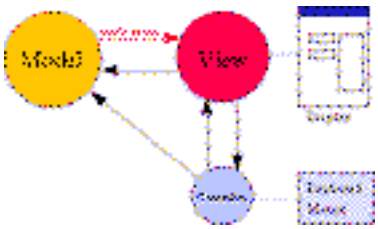


Application Controllers

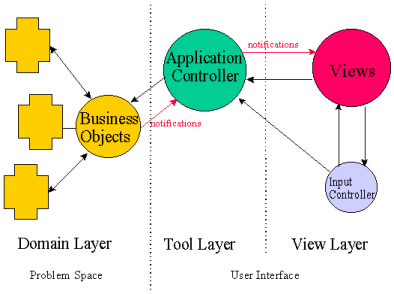
- **So far we have**
  - Action Context: data for a particular set of operations
  - Actions: code manipulating that data
- **Remember objects…**
  - Code that manipulates data belongs with that data
  - We can also use the application controller to provide context-specific action registries
  - Hold session data as well as wrap request data
- **Recall the two interpretations of MVC**

Recall MVC

Original MVC

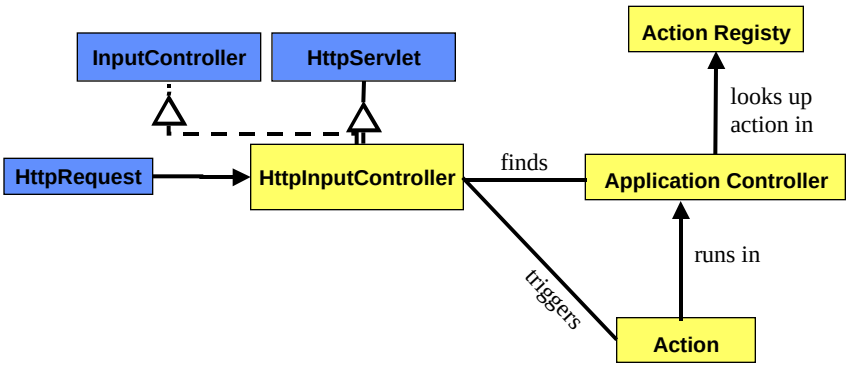


Modified MVC



Implementing Actions - Approach 2

- Extend the Context object to be a full-blown Application Controller
- Actions become methods in this controller



InputController...

```
public void service( HttpServletRequest req,
        HttpServletResponse res )
    throws ServletException, IOException {
    ApplicationController controller = this.controllerFor(req);
    controller getInputFrom(req);
    String action = controller.actionFor(req);
    controller.performAction(action);
    this.forwardPage(controller.nextPage());
}
```


ApplicationController

```
public Controller controllerFor(Request request) {
    Controller controller = (Controller)
        session.getAttribute("currentController");
    if (controller == nil) {"…set up a default…}
    return controller;
}
```

```
getInputFrom: request
"Get all of our properties that are specified in the
request"
  self properties do: [:each |
    input := request getParameter: each name.
    input isNil ifFalse: [
      self validateConvertAndSetValueOf: each to: input]].
  self validateAll.
```

Actions

Actions become methods in the application controller

```
public void showAccount() {
    account = readAccount(accountNumber());
    setNextPage("/jsp/showaccount.jsp");
}
```

We could even omit the first line, since getAccount() will do this work

```
public void withdraw() {
    status = getAccount().withdraw(getWithdrawalAmount());
    setNextPage("/jsp/reportOnTransaction.jsp");
}
```

We could even omit the first line - showAccount becomes trivial

Issues

- **Good**
  - Code and its data together in an object
  - Actions almost entirely concerned with application/business logic
  - Input parsing handled generically (requires us to define properties)
  - Better name space management
- **Bad**
  - Potential conflicts among team members working on different actions
  - Alternatives
    - *Higher-level actions with sub-actions as methods*
    - *Fine-grained controllers*
    - *Finer-grained team development tools*

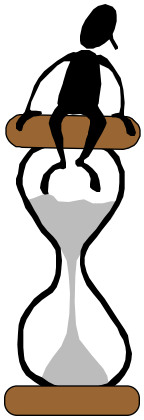
Name Space Management

- **A single client communicating wit\h multiple applications on the same application server is allocated a single HttpSession instance**
  - the applications share a common name space for user state
- **Controller or context keeps names together, reduces conflicts**
- **More flexible scoping**
  - jsp:usebean tag needs scope specified (session, request, etc.)
  - Holding state in an object lets us move information transparently between request and session

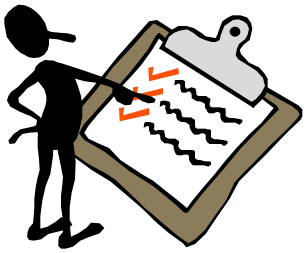
```
<% controller ct = request.getAttribute("controller");%>
<form>
    <textarea rows="3" cols="60" name="messageArea" disabled>
        <%=ct.getMessage()%>
    </textarea>
</form>
```


Performance Issues

- **Session data occupies server memory.**
- **Scalability vs. Performance**
- **Server load for high volume web applications can be reduced by analyzing the required life-span of user state and removing unnecessary items from the session.**



Form Validation



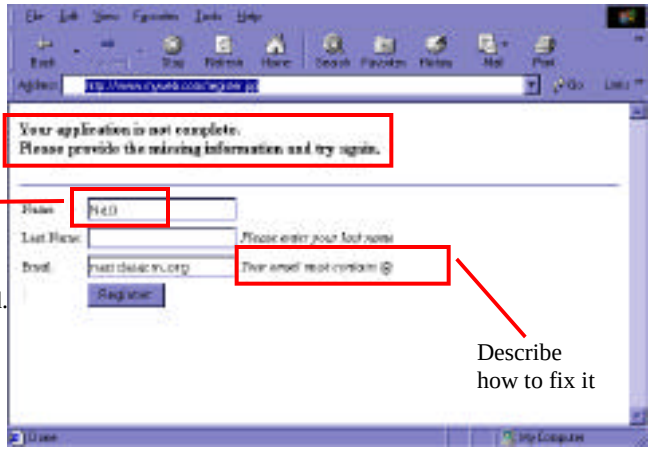
Form Validation

- Submitted information must be valid before actions can take place
- Each page is different, but the need for validation is the same
- Must tell the user how to remedy the situation
- Goal: A standard, generic way of validating information, which is easy to implement

Tell the user what you expect

You are back here because…

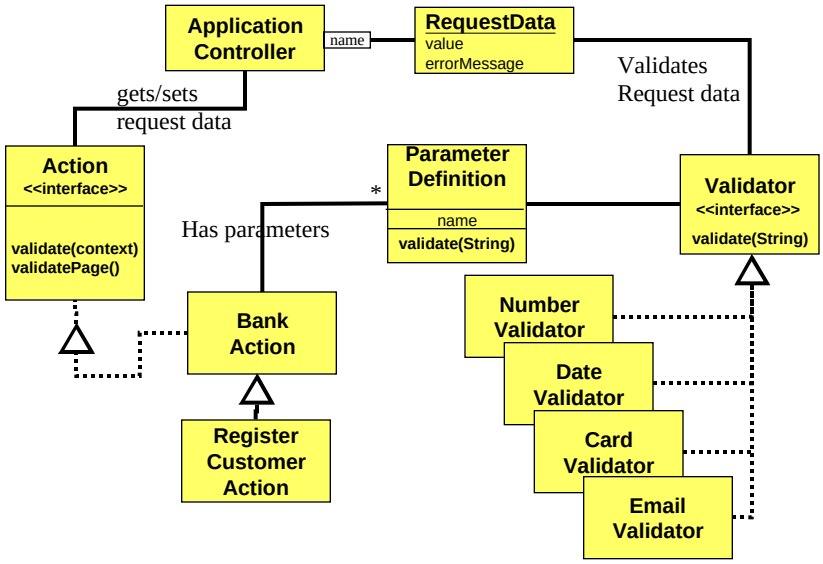
Do not discard what has been entered.



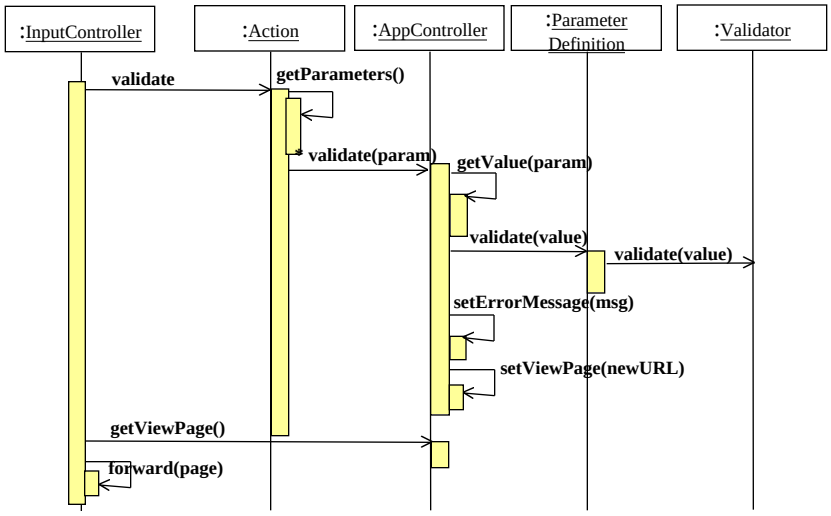
A Simple Implementation

- **Provide the rules to validate each parameter**
  - Accept and parse the value of each parameter if successful
  - Reject and provide a reason if not
- **Generalize our storage**
  - Create a class RequestData, storing either the validated value or an error message.
  - In the ApplicationController (or wherever we keep the data) store a RequestData instead of just the value.
    - *Type conversion starts to make this more complex*
  - By default all error messages are empty strings

Parameter Validation



Parameter Validation



Validators

- **Simple objects that perform data entry validation**
  - Business rules and data correctness at field level
  - Also need page level, transaction level validators

```
public class EmailValidator implements Validator {
public String validate(String value) {
    //a@b.com
    if(value !=null && value.length()>=7 && value.indexOf("@") >=0)
        return null;
    return "Emails must be like a@b.com";
    }
}
```


InputController

- **Input controller can initiate validation**
  - The next page will be set to either the current page (with errors) or the next valid page
  - In previous processing, the values have been extracted and set into the applicationController

```
public void triggerAction(Action action,
  HttpServletRequest req, HttpServletResponse res){
    if(action.validate())
        action.trigger();
    forwardPage(action.controller().getViewPage(),req,res);
}
```

Action

```
validate
    valid := parameters allSatisfy: [:each |
        appController validate: each].
    valid ifFalse: [
        appController viewPage: self validationPage].
    ^valid.
```

- Check that all parameters pass, and set the view page accordingly
- Actions start to become more like page definitions

ApplicationController...

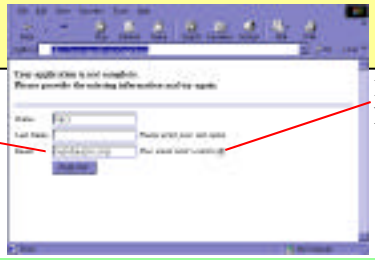
```
public boolean validateParameter(ParameterDefinition param) {
    String name = param.getName();
    String error = param.validate(this.getValue(name));
    if (error != null)
        setErrorMessage(name, error)
}
```

Server Page

```
<jsp:useBean id="controller"
      scope="request" class="MyAppController"> </jsp:useBean>

..
<input type="text" name="email"
    value="<%= controller getValue: 'email'%>" >
    <I><%= controller errorMessageFor: 'email'%> </I>
..
```

This will create a new one if there is none



Do not discard what has been entered.

Describe how to fix it

Formatter Objects Revisited

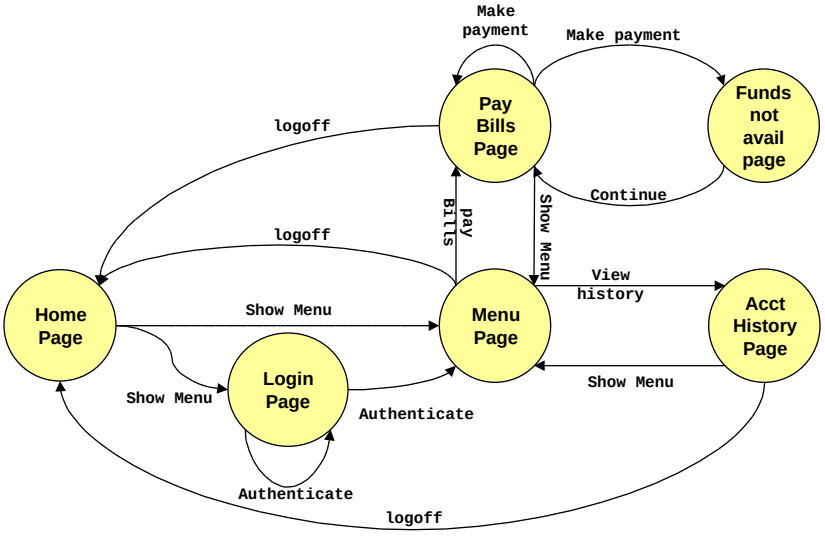
```
<jsp:useBean id="controller"
      scope="request" class="MyAppController"> </jsp:useBean>
<jsp:useBean id="inputText"
       scope="application" class="InputText"> </jsp:useBean>
..
Name: <%= inputText show: 'name' in: controller. %>
Last Name: <%= inputText show: 'lastname' in: controller. %>
Email: <%= controller showInputTextFor: 'email'%> //other form
..
```

Let the Bean/Taglib to do the work

Flow Management

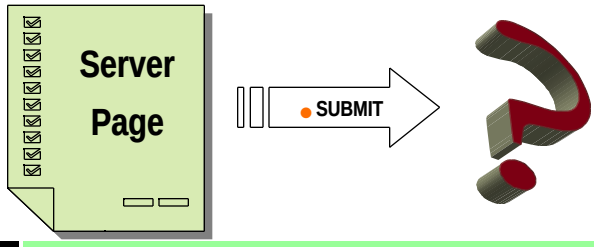


Website "Expected" Flow Diagram



Flow is interesting

- **What do you flow from?**
  - JSPs
  - Business (Model) behavior Invocation
- **What is Flow?**
  - The transition between these states



State Machines

- **We can think of this as a finite state machine… or not**
- **Pages represent state**
- **Transitions initiated by user actions**
  - Certain transitions may be impossible, or have preconditions
- **How rigid do we want to be?**
  - Maintaining application consistency
  - vs. Annoying the user
- **Transitions define a finite state machine**
  - *Implicit - Simpler*
  - *Explicit - More Rigorous, Declarative*
  - *Partial*

Flow Management

- **Declarative Approach Benefits:**
  - decoupling of site navigation from code
  - possible tool support for navigation planning
  - navigational changes without code changes
- **Approaches**
  - Ad Hoc
  - Flow Policies
  - Event-Listener (Observer pattern, but in a non-MVC context)
  - Full state machine
  - "Seaside" (continuation-based)

You cannot always dictate incoming requests
"Back" button, deep links

Flow Management: Ad Hoc



Flow Management: Ad Hoc

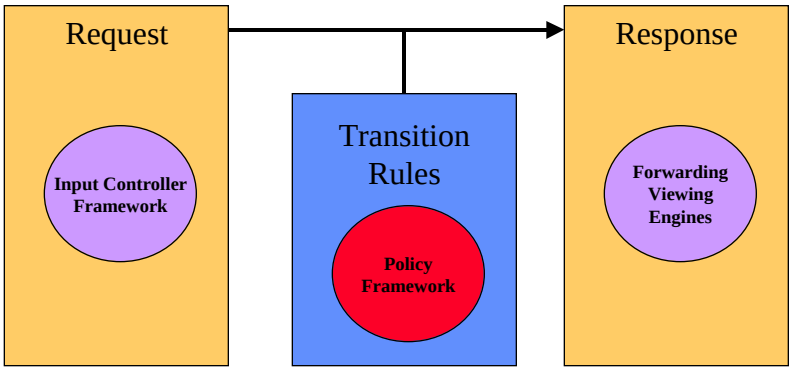
- **In other words, the normal thing**
- **State transitions hard-coded into pages, actions, or application controllers**
  - getNextPage
- **Advantages**
  - Simplest thing that could possibly work
  - All that's necessary for lots of applications
- **Disadvantages**
  - Hard to manage complex transitions
  - Hard to change
  - State transitions tied in with other logic

Flow Management: Policies



Flow Policy

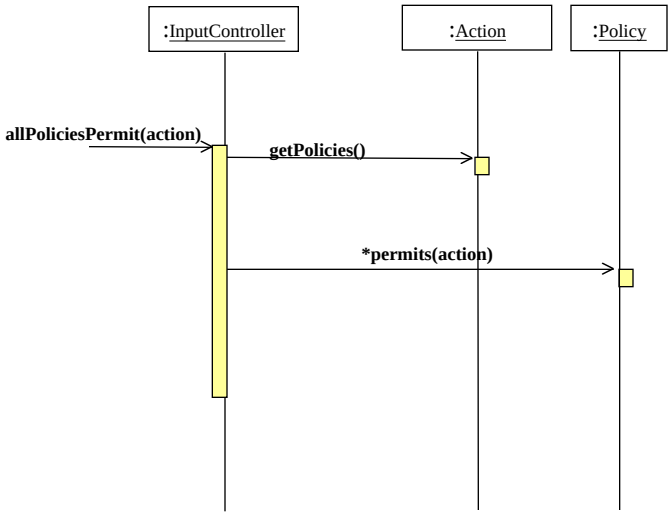
Lightweight framework to guarantee pre-requisites are met.



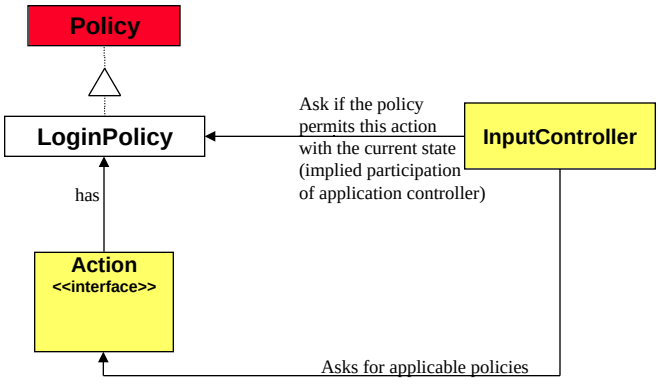
Policy

- **Not as complicated as a state-machine**
- **Best for applying pre-requisites**
  - Needs Form Validation
  - Needs SSL
  - Must be logged in
  - Must accept license agreement
  - etc.

Policy...



Example: LoginPolicy



Checking Policies

- **Input Controller checks all policies before triggering action**

```
triggerAction: action request: request response: response
    (self allPoliciesPermit: action)
        ifTrue: [action trigger].
    self
        forward: action
        request: request
        response: response.
```

If the policies are succesful

Implications

- **Light-weight and easy to implement**
- **Works well for broad rules (e.g. must be logged in, cannot go backwards)**
- **Not as good for detailed site-navigation rules**
- **Simple procedure to add transition rules to actions**
  - Can be declarative or stored in files

Flow Management: Events

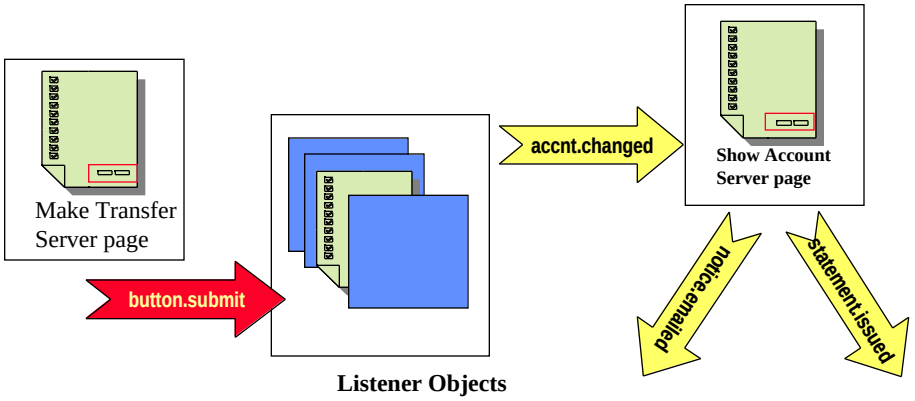


Event Model

- **Similar to traditional GUI-based Event Controllers**
  - There are objects that generate events
  - There are listeners which act on these events
  - States tell how they can be reached (register as listeners)
- **Events**
  - Submit events (buttons)
  - Links
- **Application Events**
  - Arbitrary events generated by the application

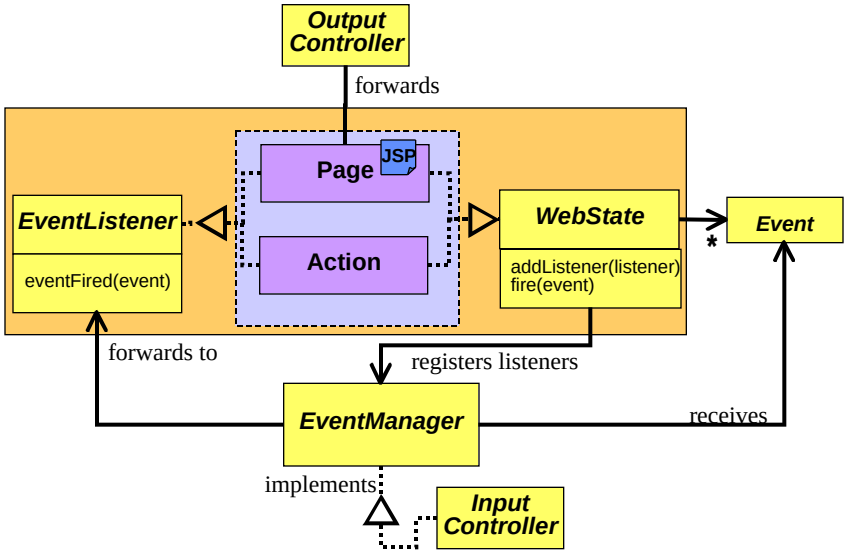
Site flows toward listeners responding to events
The website is a chain of events and listeners

Event Model



Pages respond by showing themselves!
There is only one http response (Page bundles/Portals)

Web Event Engine



Special Considerations

- **Server Pages**
  - Lazy initialized –
    - *They become Observers before they are instantiated!*
    - *Proxy Pages can help – They forward to the page when events are fired*
  - HTML cannot fire events- Need special tags to identify events
- **Mapping events**
  - We need a “registry” to keep track of the Observer relationships at a meta level
- **Entry Points**
  - If states can only be reached via the events they listen to we need to start the system in a uniform fashion

JSP Page with Event Tags

- **Only forms that submit and Links generate interesting events**
- **It is possible to find naming standards to deal with many events**
  - <namespace>.<origin>.<event>

```
<a href="/myapp/help>Need help?</a>
..
<form action="/mayapp/register" method="post">
</form>
```

These will tell us the events

Action firing event

- No need for setting the “nextPage”
- Fires an application event after it’s task

```
public void trigger(ActionContext context) {
    ..
    this.fireEvent(this.BILLPAIDEVENT);
}
```


Event Map

```
<observer id=HOMEPAGE uri=/home.jsp />

<observer id=PAYMENTPAGE uri=/paybills.jsp />

<observer id=CONFIRMPAGE uri=/confirm.jsp>
       <listen event=billpaid source=PAYMENTACTION>
</observer>

<observer id=PAYMENTACTION class=PayBillAction>
   <listen event=paybill source=PAYMENTPAGE>
</observer>
```

- Can use a simple map of events
  - Parse and keep it memory
  - InputController handles event dispatches

How to implement...

```
public class InputController .. implements EventManager {
    public void dispatch( ActionContext context ) {
    Event event = context.getEvent();
    Iterator iterator = this.getEventListeners(event);
    while (iter.hasNext()) {
        EventListener listener=(EventListener) iter.next();
        listener.eventFired(event, context);
        }
    }
}
```

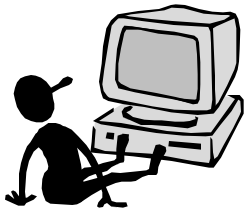
```
public class Page .. implements EventListener {
    public void eventFired(Event e, ActionContext context) {
        context.forward(this);
    }
}
```

```
public class Action .. implements EventListener {
    public void eventFired(Event e, ActionContext context) {
        this.trigger(this, context);
    }
}
```


Event Issues

- **Explicit representation for web-site state transitions.**
- **Simple, powerful and intuitive model**
- **Can be integrated with other features of InputControllers**
  - Validation, etc.
- **Possible conflicts with many pages responding to the same event**
  - Requires tool support or careful validation
- **Name space problems**
  - Many application and page events

Flow Management: State Machine



Explicit State Machine

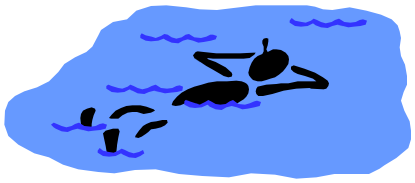
- Define explicit objects representing states and transitions and a transition table
- Form submission goes through a state machine servlet that invokes the state machine machinery
- Can be defined by state and transition tables plus Transition subclasses

```
MAINMENU=EnsureLoggedInTransition.class,MENUSTATE,LOGINSTATE
LOGIN=AuthenticateTransition.class,MENUSTATE,LOGINSTATE
PAYBILL=EnsureLoggedInTransition.class,PAYBILLSTATE,LOGINSTATE
VIEW= EnsureLoggedInTransition.class,VIEWACCTSTATE,LOGINSTATE
etc.
```

Implications

- **Complex System**
  - Significantly harder to understand/debug
  - Strict rules make flexible navigation harder
  - Users may feel like they are in straight-jacket
- **Site navigation is decoupled from code**
- **Site navigation is explicitly defined in a single location**
  - lends itself to tool support
- **May be easier to integrate standardized validation, error handling**
  - enforces strict coding
- **Can quickly mockup a site and add business logic incrementally as the transitions are implemented**

Flow Management: Seaside



Flow Management: A Radical Approach

- **We have passively accepted the web's reversal of "normal" algorithmic computing. Instead, let's impose our point of view on the web programming.**
- **Seaside: A Continuation-based web flow framework**
  - http://sourceforge.net/projects/seaside/
  - (it does other stuff too, this is just the most interesting aspect to me)
- **Open source framework, based on Squeak (www.squeak.org)**
- **Turns around the programming model**
  - We issue a series of requests to the user for information
  - Requests return with form data, etc.
  - self getLoginInformation.
  - Nice, but how?!?!

Continuations

- **Continuations are a functional language construct related to function return, and to goto.**
- **Makes the idea of a stack, and returning from a stack, first class.**
- **Within a method I can return.**
  - Suppose I could capture the state, and do it more than once

```
answer
   ^2 + 2.
answerFor: x
   ^2 + x
thingThatCalculatesAnswerFor: x
   ^[^2 + x]
```

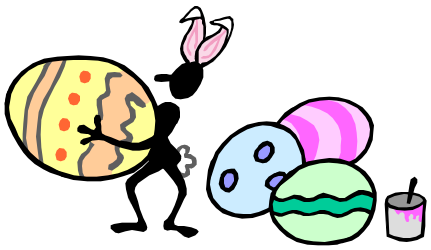
Continuations (cont'd)

- **So a method can return more than once (!)**
- **Steps**
  - We invoke a method like getLoginInformation
  - Page is sent to the user, and our method blocks
  - When the page is submitted, our method invokes the continuation
- **Suppose the user hits the back button and submits an earlier page.**
  - An earlier continuation is invoked.
- **Tricky concepts**

Seaside

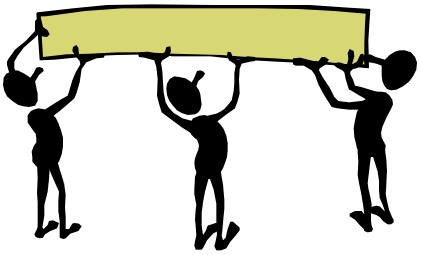
- **Advantages**
  - Very natural control flow
  - Nice handling of state transitions
  - Good page reuse: the same page submit can take you to different places, depending how it was invoked.
- **Disadvantages**
  - Very strange internals
  - Non-standard and "weird"
  - Scalability? Each page needs a copy of the stack = memory
  - Young and relatively untested
- **Even if I'd be unlikely to use this in practice, it broadens my understanding of what's possible. Very "blue plane"**

Real-World Usage



Real-World Usages

- **Lots of general patterns. What real-world frameworks use/illustrate some of these ideas.**
  - What. The stuff we do with clients isn't "real"? ☺
- **Lots (and lots) of frameworks out there**
- **Two examples**
  - Apache Struts
  - Oracle "Project Cleveland"



Struts

- **Open-source framework from Apache group (Jakarta)**
  - example of Sun's "model 2 web programming"
- **JSP-based, with tag library**
- **Light-medium weight**
- **Ideas**
  - divisions: input, actions, presentation, and model
  - single input controller servlet
  - no application controller: part of model, or actions, or declarative
  - state transitions controlled by configuration file
  - model divided into "bean forms" (state only) and normal objects

Project Cleveland

- **Oracle 9iAS MVC Framework**
  - Available from Oracle. I believe tied to their app server.
- **Heavyweight, oriented towards tool support and automation**
  - Generation of components from UML
  - "little or no java coding required to create a working application"
- **Layer on top of J2EE**
  - "Isolates from the various and ever-changing J2EE APIs"
  - Abstractions for web, components, persistence, etc.
    - *Spans JDBC, TOPLink, BC4J, etc.*

Seaside

- **Recall the seaside framework we discussed earlier**
- **Server-pages with custom mechanism (pattern matching in names)**
- **Lightweight (but tied to its own http server)**
- **Ideas**
  - Control flow via continuations
  - Automated input controller to model mapping (based on names)
  - Emphasis on re-use of templated pieces
  - “transactions” to control groups of pages



Summing Up



Summary

- **Good structure is important**
- **Web technologies have many traps, but they can be dealt with**
- **Ideas**
  - Separation of Concerns
  - Once and Only Once
  - Debuggable and Testable code

Summary -- Specific Ideas

- **Model-View-Controller**
  - Many interpretations
  - Input Controller vs. Application Controller
  - Input Servlet -> Presentation Server Page
- **Action Framework**
  - Single Input Servlet
  - Make Actions to reduce the amount done by servlets
  - Move state into User Context or Application Controller
  - Keep business rules in domain objects
- **Multiple Front Ends**
  - Can further separate input and output controllers to support multiple presentation and input technologies

Summary -- Specific Ideas (2)

- **Form Validation**
  - define the parameters and their validation
  - allow the infrastructure layer to do validation automatically
- **State Transitions**
  - Many different approaches
  - different degrees of complexity
  - different degrees of explicitness

References

- Objects and the Web
  - Alan Knight , Naci Dai, IEEE Software, March-April 2002
- Best Practices for Developing Web Applications Using Java Servlets
  - Michael Ellis, Naci Dai, OOPSLA 2000
- Integrating Business Objects into J2EE
  - Martin Fowler, Alan Knight, Kai Yu, Java One 2000
- Martin Fowler: Patterns of Enteprise Architecture (2002)

Acknowledgements

- **Some pictures from**
  - BEA training materials
  - Object-Arts (Dolphin Smalltalk) web site
- **Ideas**
  - Lots of people, but most directly
  - Michael Ellis (WebGain)
  - Martin Fowler (ThoughtWorks)

Contact Information

- **Naci Dai**
  - Corporate Slave
  - nacidai@acm.org
- **Alan Knight**
  - Smalltalk Diehard
  - knight@acm.org

Thank You!



- www.smalltalkchronicles.net/objectsvsweb/slidesNOD.pdf