the POWER of JAVA™

Java™ EE 5 BluePrints for AJAX-Enabled Web 2.0 Applications

Sean Brydon, Greg Murray, Inderjeet Singh, Mark Basler

Java BluePrints
Sun Microsystems, Inc.
http://blueprints.dev.java.net/

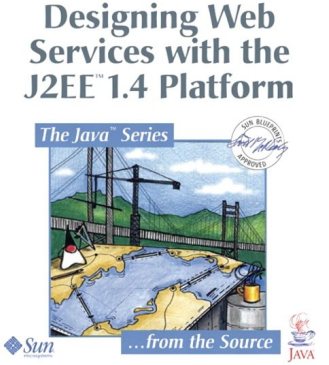
TS-1615

Goal of Our Talk

Learn how to architect and build AJAX-enabled Web 2.0 applications using Java Enterprise Edition™ (Java EE™) 5 platform

Speaker's Qualifications

- Members of the Java BluePrints Program at Sun Microsystems
  - http://blueprints.dev.java.net/
  - Programming Model, Guidelines, Patterns
  - Open-Source BSD License Projects
  - Projects:
    - Java Pet Store, *a new version showing Web 2.0 with Java EE 5*
    - Java BluePrints Solutions Catalog
    - Java Adventure Builder
  - Books
    - Designing Web Services with the J2EE 1.4 Platform
    - Designing Enterprise Applications with the J2EE Platform, 2nd Ed



Web 2.0: Salient Aspects

- Web as a Platform
  - Network is the computer
  - Lightweight de-facto programming models
    - For example: SOAP vs. REST
  - Mashups

- Richer User Experience
- Community created content
  - Collective Intelligence for collaborative categorization
- New security issues
- Do you have a long tail?
- BluePrints focussed on engineering aspects

Agenda

Brief AJAX Overview

Demo

Application Design

JavaScript Guidelines

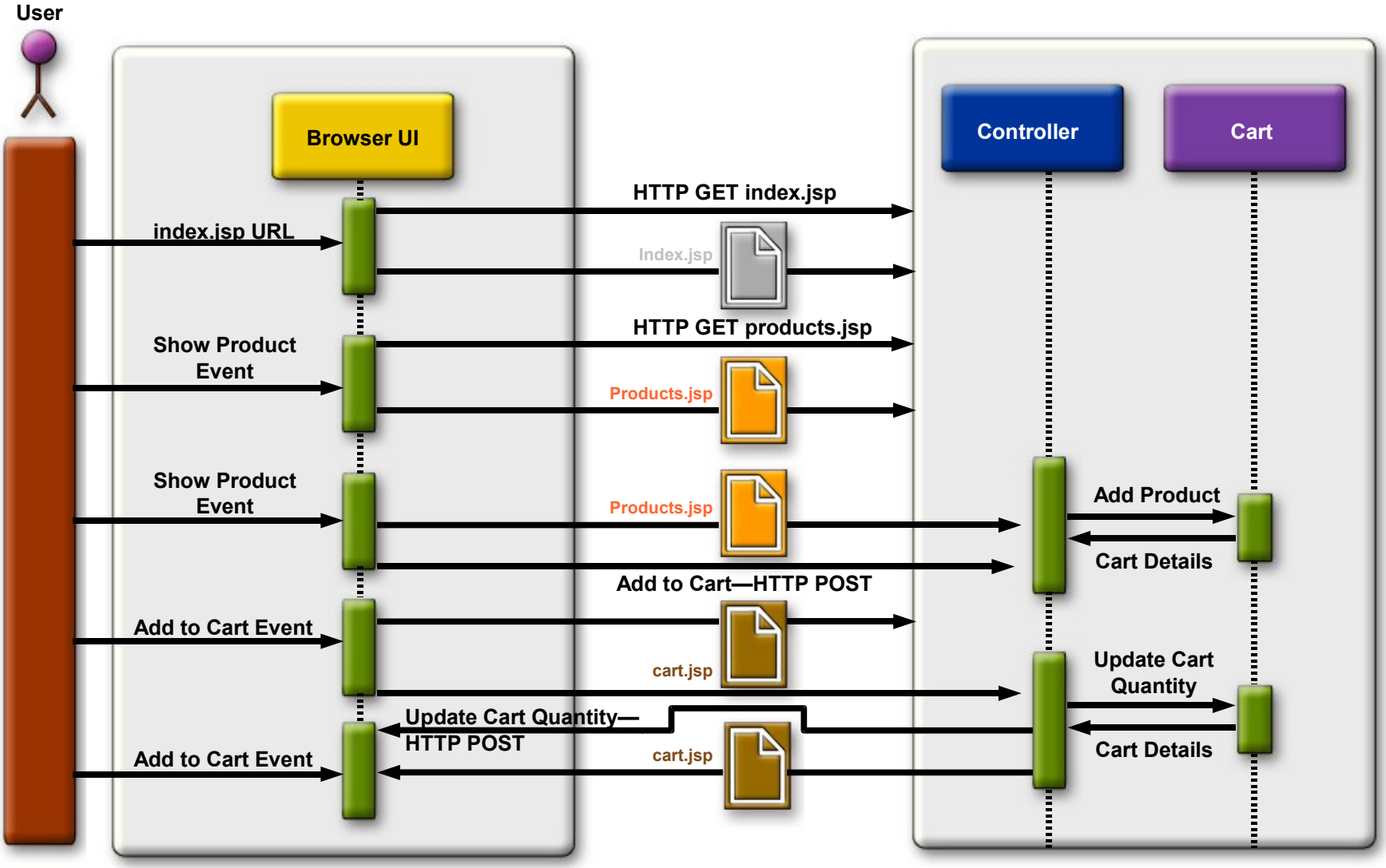
JSF Approach

Summary

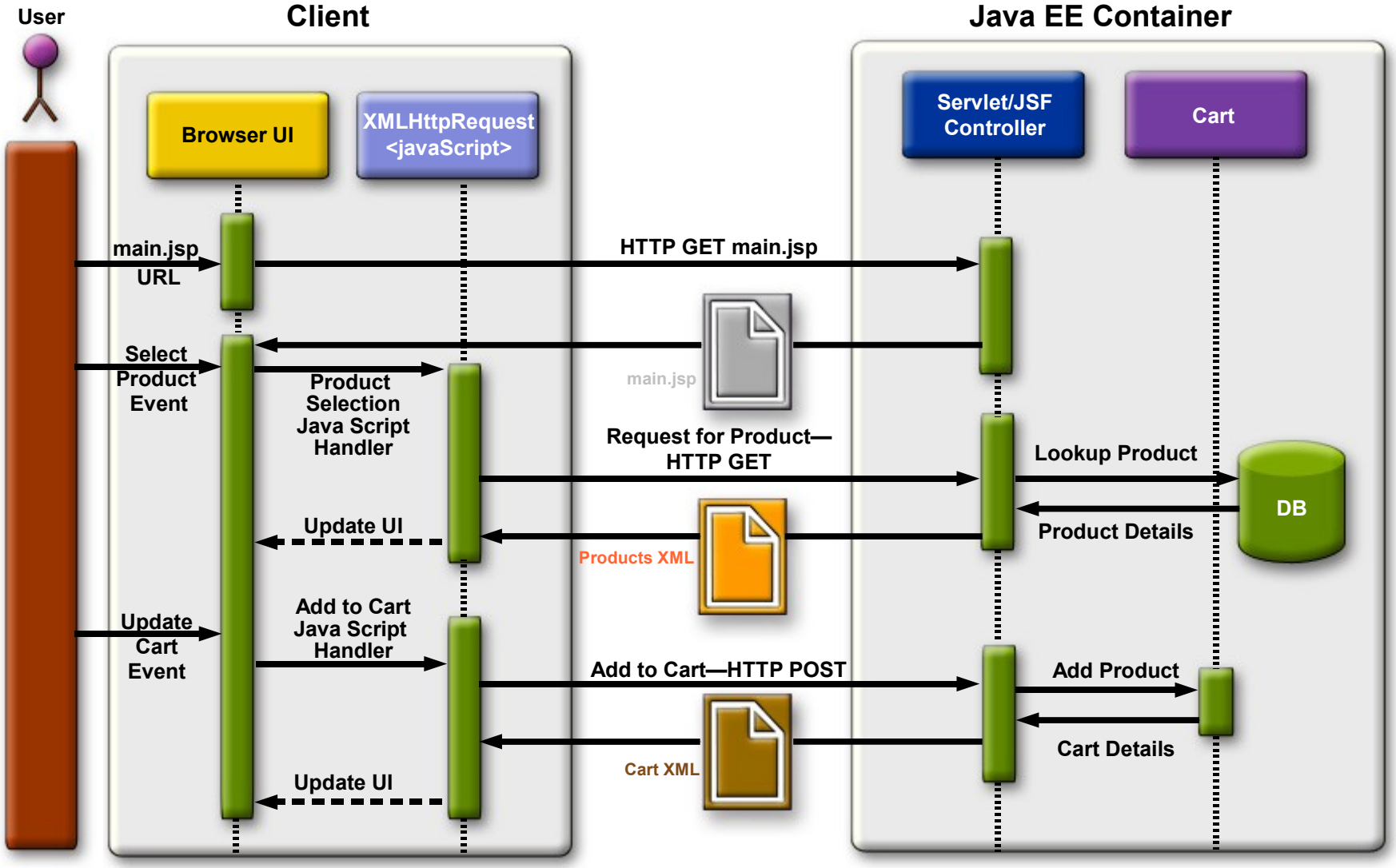
Conventional vs. Rich Web Applications

- Conventional Web Applications
  - Server Centric
  - Page to Page navigation based
- Rich AJAX Web Applications
  - Client executes logic
  - Client holds some data
  - Page is the application

Conventional Interaction Model



High-Level AJAX Interaction Model



Simple AJAX Example

```
var req;
function doCompletion() {
    var target = document.getElementById("autocompleteField");
    var url = "autocomplete?action=complete&id=" +
encodeURI(target.value);
    req = new XMLHttpRequest();
    req.onreadystatechange = myCallBack;
    req.open("GET", url, true);
    req.send(null);
}
function myCallBack() {
  if (req.readyState == 4) {
     var resp = req.responseXML;
     //get data from XML doc
     var personName = resp.getElementsByTagName("person")[0];
     //update dom to add name to page
     mydiv = document.getElementById("peopleListID");
     mydiv.appendChild(document.createTextNode(personName));
   }
}
```

DEMO


Java Pet Store
Seller | Search | Catalog | Map | Main
News from BluePrints
SystemNews discusses Java BluePrints AJAX Components
Dogs
Cats
Birds
Fish
Reptiles

Agenda

AJAX Overview

Demo

Application Design

JavaScript Guidelines

JSF Approach

Summary

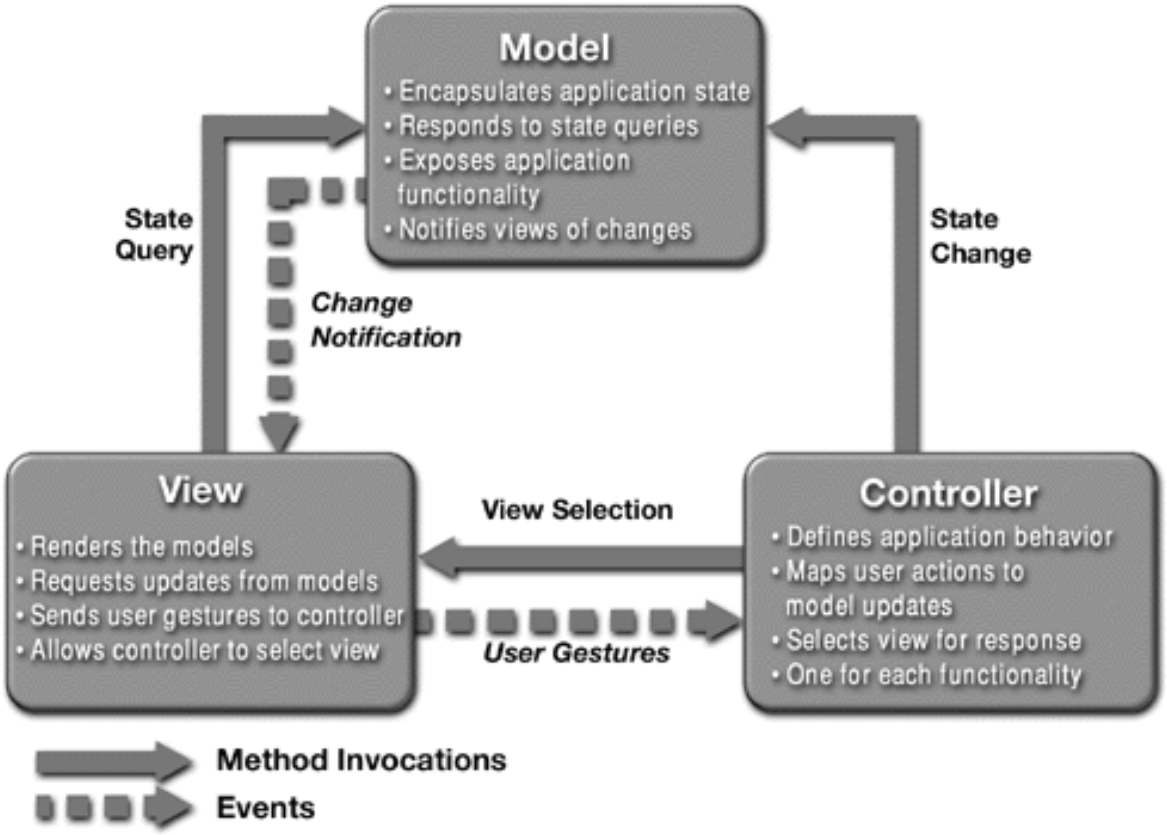
Pet Store 2.0 Design Choices

- When to Use AJAX
- Page is the Application Architecture
- Model View Controller and Patterns
- Leverage Existing AJAX Libraries
- Use JSF Components to Wrap AJAX
- Mashup Architectures
  - Proxy for cross domain
  - Rest service APIs
- Domain model to store and manage data
  - Including user content and images
- Now Build it on Java EE 5!

AJAX Design Choices

- When to Use AJAX
  - When it enhances user experience
  - “But it’s fun to go crazy!”, tech team quote
- Page is the Application
  - Really just one page?
  - Client and server split MVC responsibilities
- Server Centric
  - Server renders everything
- Client Centric
  - More logic coded in JavaScript
  - Client controller
  - Consider existing AJAX library design

Model View Controller



Split Model View Controller

- Client and Server Code need MVC
- Model
  - Java Persistence APIs for Domain Model
  - XML/JSON for Model Data Transport
  - Programmatically Cache Locally in Client
- Controller
  - Server-side is a Servlet or JSF component
  - Server-side returns JavaScript
  - JavaScript executed on client
- View
  - Server Returns Fragments and Style Sheets
  - Presentation Handled by Browser

New Security Issues

- Upload user content
  - Programmatic validation
  - User policing of content—no naked pets!
- CAPTCHA
  - Avoid automated graffiti
- JavaScript Sandbox
  - Origin of domain policy for executing code
- We accessed an RSS over HTTPS
- Using REST APIs
  - Trust Google code on your clients?
- JavaScript code visible to the world
- HTTP-->HTTPS—Requires a page refresh

Mash Design Choices

- Pet Store Uses Four Services
  - Google Map, Yahoo GeoCoder, Pay Pal, RSS feed
- Client directly
- Proxy
- Feeling Clean?
  - Don’t need SOAP!
  - Don’t need WS-Infinity
  - Take a REST
- Wrap in JSF component

Proxy for Cross-Domain

- Example: News Bar using RSS Feed
- Client uses server to mediate with service
- Avoid Server of Origin Security Policy
- Why?
  - Mitigate slow RSS over HTTPS
  - Server pre-processes data
  - Read RSS feed, similar to a datasource
  - Parse big document
  - Return as JSON
- Application-Scoped Data so Cached
- Security Settings on Server

Client-Side Mash Up

- Example: Google Maps for Pet Sale Search
- Use Third-Party Service APIs from Client
  - Presentation and logic comes from service
- Client-Specific Data
- Existing API Satisfies Need
- Hard to Achieve with Proxy Style
  - Handle Presentation Code
- Give up some control

Client-Side Mashup

- Server of origin policy
  - Can not just XMLHttpRequest from page
- Script is Loaded From Third-Party
- Client Request Fetches Page Which Includes Google JavaScript
- Third-Party Code Is Executed in Client

```
<script type="text/javascript"
src="http://maps.google.com/maps?file=api&v=1&key=ABI...">
</script>
<script type="text/javascript"
            src="/petstore/faces//mapviewer/script.js">
</script>
```

Model Tier

- Does Model Need to be Different for AJAX apps?
- Use Java Persistence APIs for Domain Model
  - Goodbye EJB CMPs!
- Use Facade Pattern
  - Web object or Session Bean
  - Transactions and entity manager access encapsulated
  - Detached objects returned from client of facade
- Keep Transformation Code Separate
  - Model is POJOs
  - Client expects XML, JSON, HTML, text, JavaScript code

Use Java Persitence APIs

```
@NamedQuery(
  name="Item.getItemsPerProductCategory",
  query="SELECT i FROM Item i WHERE i.pID = :pID")
@Entity
public class Item implements java.io.Serializable{
  private String itemID;
  private String productID;  //other fields ...
  public Item() {}

  @TableGenerator(name="ITEM_ID_GEN",table="ID_GEN"...)
  @GeneratedValue(strategy=GenerationType.TABLE,
                          generator="ITEM_ID_GEN")
  @Id
  public String getItemID() {
    return itemID;
  }
//other getters & setters...
```

Use Model Facade Pattern

```
public class CatalogFacade
                implements ServletContextListener {
  @PersistenceUnit(unitName="PetstorePu")
        private EntityManagerFactory emf;

  @ResourceUserTransaction utx;
  ...

  public List<Item> getItems(String pID){
    EntityManager em = emf.createEntityManager();
    Query query =   em.createNamedQuery
                    ("Item.getItemsPerProductCategory");
    List<Item>  items =
          query.setParameter("pID",pID).getResultList();
    em.close();
    return items;
  }
```

Agenda

AJAX Overview

Demo

Application Design

JavaScript Guidelines

JSF Approach

Summary

AJAX Guidelines

- JavaScript Libraries
- Eventing
- Return content-types
- Value List Handler
- Usability
- Mashups

JavaScript Programming Language Libraries

- Prototype
- RICO
- Script.aculo.us
- Dojo
- Zimbra

Recommendation: Adopt a library and don't try to re-invent the wheel.

Eventing

- Anonymous
- Publish/Subscribe
- Closely coupled

Recommendation: Consider the meaning of each and weigh the benefits when designing your application.

Eventing

Java Pet Store

Seller | Search | Catalog | Map | Main

News from BluePrints Free Movies of Re-born Pet Store 2.0

Pets

Cats

Dogs

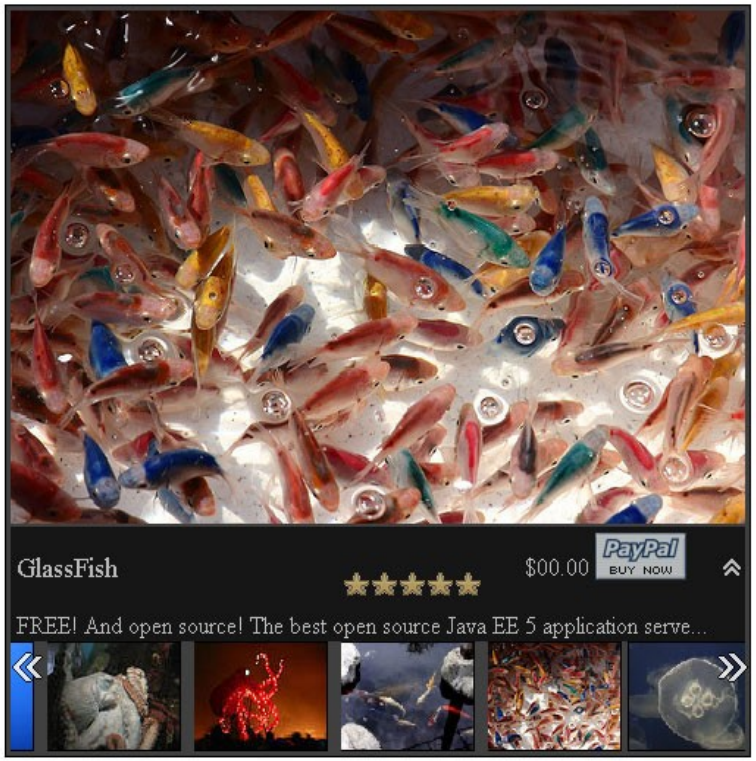
Birds

Reptiles

Fish

Small Fish

Large Fish



JavaScript and AJAX Conventions Used in Petstore Architecture

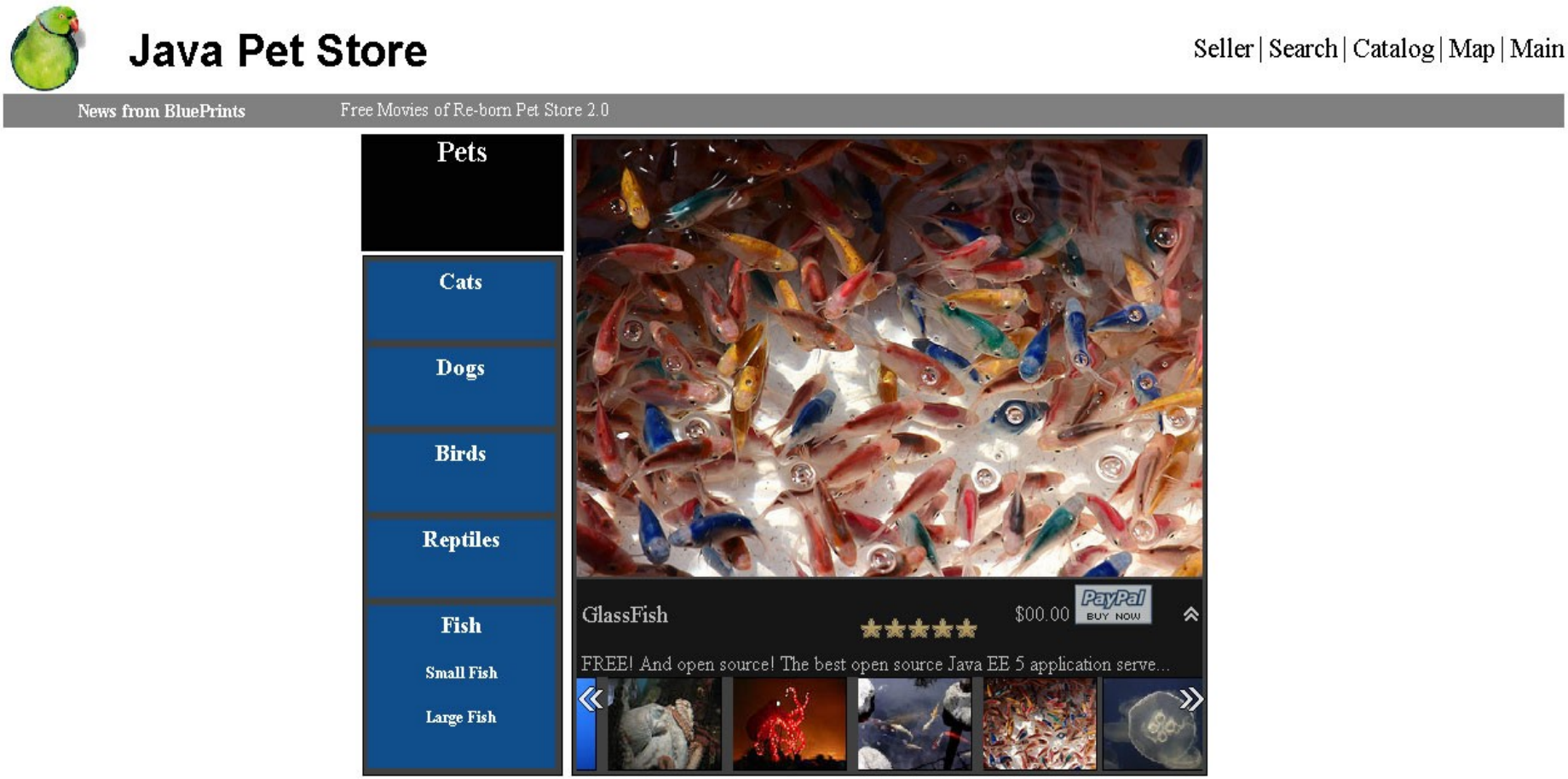
- Standardized on Dojo
  - dojo.event.connect
  - dojo.io.bind
  - dojo.widget.*
- Used Object-Oriented JavaScript
- Cleanly Separation of CSS/JS/HTML (MVC)
- Used Namespaces as much as possible
- Used feature detection
- Favored DOM injection over innerHTML

Response Content Type

- XML
- HTML
- Text
  - Post processing on client
  - Inject directly into the page
- JavaScript
  - Evaluated in JavaScript using eval()
  - JavaScript object representations of data(JSON)

Recommendation: Use XML for structured portable data. Use plain text for when injecting content into the HTML. Use JavaScript to return object representations data.

Value List Handler



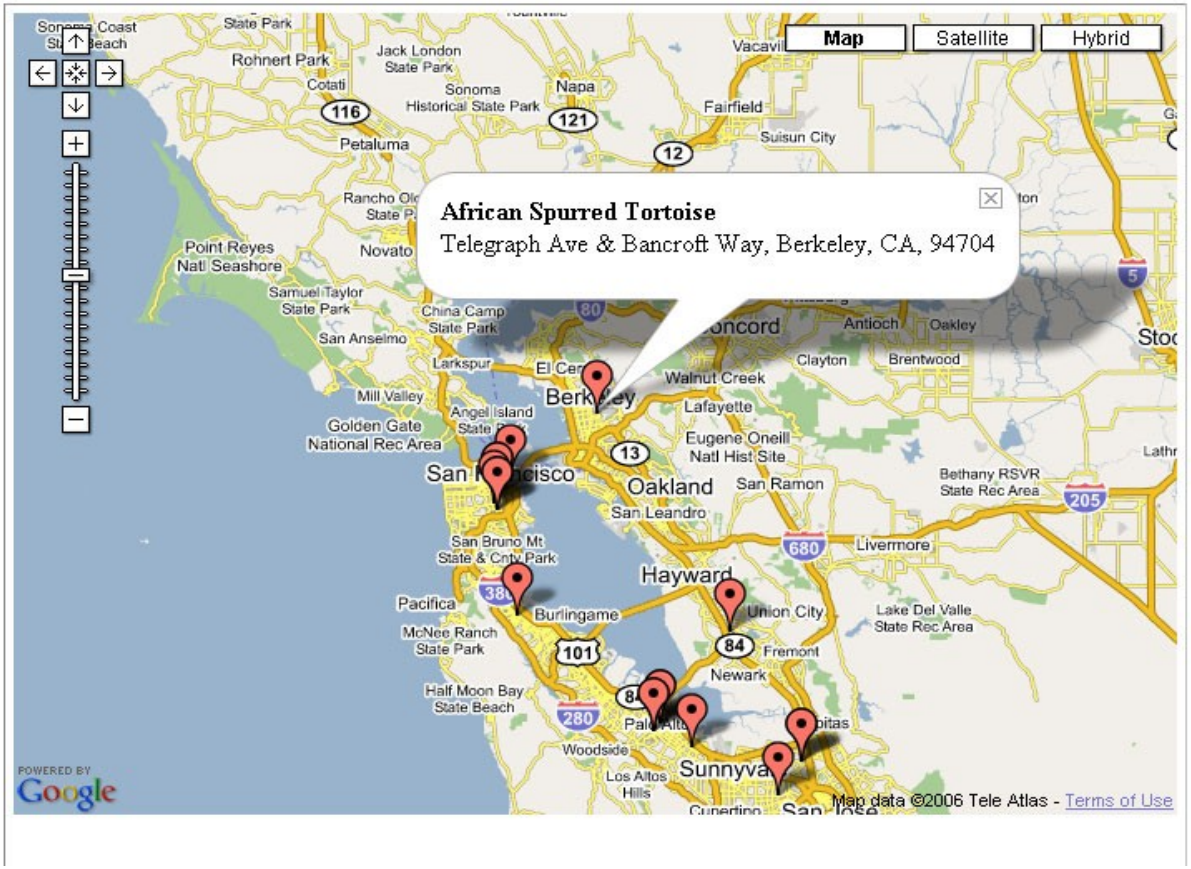
What About Usability?

- Integrated JSF components with their own packaged JavaScript
- Externalized customizable portions of a component rather than embed it in JavaScript (Info Pane)
- Bookmarking using the # (anchor technique)
- Exposing access to specific chunks of data via the CatalogFacade.

Petstore Mashup

19 Items Displayed

- **African Spurred Tortoise** *(detail)*
  Telegraph Ave & Bancroft Way, Berkeley, CA, 94704
- **African Spurred Tortoise** *(detail)*
  River Oaks Pky & Village Center Dr, San Jose, CA, 95134
- **Box Turtle** *(detail)*
  San Antonio Rd & Middlefield Rd, Palo Alto, CA, 94303
- **California Desert Tortoise** *(detail)*
  Millbrae Ave & Willow Ave, Millbrae , CA, 94030
- **Florida King Snake** *(detail)*
  Paseo Padre Pky & Fremont Blvd, Fremont, CA, 94555
- **Frog** *(detail)*
  University Ave & Middlefield Rd, Palo Alto, CA, 94301
- **Green Iguana** *(detail)*
  Leavesley Rd & Monterey Rd, Gilroy, CA, 95020
- **Green slider** *(detail)*
  Palm Dr & Arboretum Rd, Stanford, CA, 94305
- **Guinea Pig** *(detail)*
  Dolores St & San Jose Ave, San Francisco, CA, 94110
- **Hawaiian Green Gecko** *(detail)*
  20th St & Dolores St, San Francisco, CA, 94114
- **Iguana** *(detail)*
  Cesar Chavez St & Sanchez, San Francisco, CA, 94131
- **Iron Dragon** *(detail)*
  Campbell St & Riverside Ave, Santa Cruz, CA, 95060
- **Large Box Turtle** *(detail)*



Mashups From a JavaScript Perspective

- Expose data via JSON for a data-centric mashup
  - Allow for a callback function
- Use document.write() if you want to prevent others from re-purposing your mashup data
- Consider using a key passed in as a URL parameter to prevent improper use
- Consider restricting access to a list of hosts
- Consider Dojo ScriptSrcIO

Agenda

AJAX Overview

Demo

Application Design

JavaScript Guidelines

JSF and AJAX

Summary

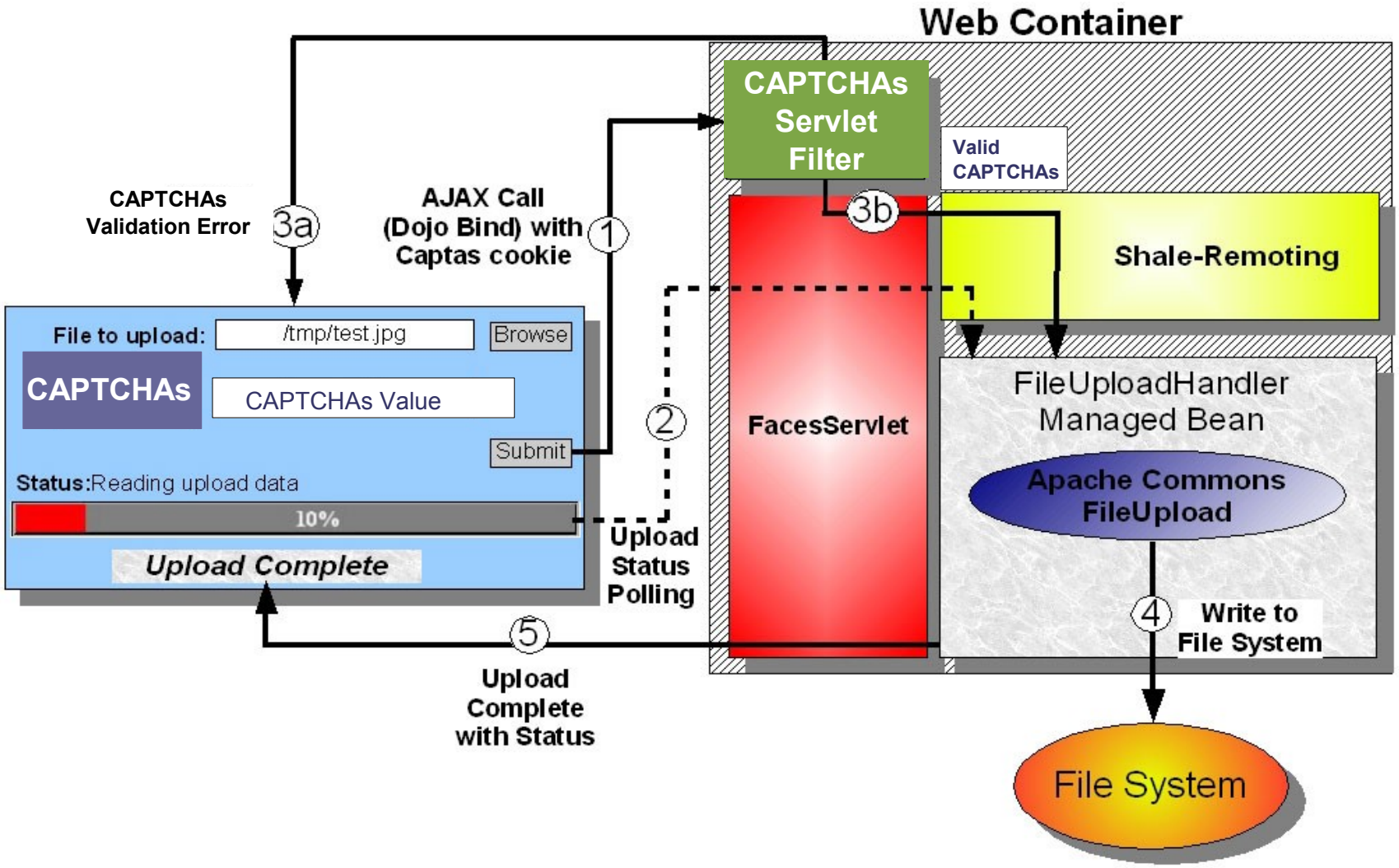
JSF Approach—General

- Took a hybrid approach
- Goals
  - Capture a wider audience
  - Zero setup
- Encapsulates an AJAX component
- Client (browser) can share some of the load for more flexibility
- Demonstrated Javascript frameworks and their widgets
- Still exploring alternative approaches

File Upload Component Implementation

- File Upload through AJAX
- Utilized open source libraries
- Progress bar status
- Ongoing status Available
- CAPTCHAs
- Store the data
- Extending component

File Upload Component's Submit



Accessing Static and Dynamic Resources

Directly Accessing Resources

- Traditional Web Design
- Artifacts need deployment descriptor
- Possible name clash
- Could be appropriate for small groups

Accessing Static and Dynamic Resources

Renderer Serve Resources

- Serve component's resources through renderer
- During the "Apply Request Values" phase
- Serve static resource
- Execute/delegate dynamic call
- Call ResponseComplete method

Accessing Static and Dynamic Resources

Renderer Server Resources (disadvantages)

- “Restore View” phase has to reconstitute the component tree
- Performance Consequences
- Especially if state is maintained on the client
- Phase has to finish executing
- Side-effects also with "immediate=true"
- Not the optimal approach

Accessing Static and Dynamic Resources

PhaseListener Serves Resources

- During the “Restore View” phase
- Serve static resource
- Execute/delegate dynamic call
- Call ResponseComplete method

Accessing Static and Dynamic Resources

PhaseListener Server Resources (disadvantages)

- PhaseListener for each component
- Developers coding differently
- All PhaseListeners fired sequentially
- Performance burden

Accessing Static and Dynamic Resources

Third-Party Libraries (Shale-Remoting)

- Keeps developers from adding functionality
- Single PhaseListener
- Doesn’t require developer configuration
- Serves static requests
- Delegates dynamic requests
- URI starts with context root

Accessing Static Resources

```
import org.apache.shale.remoting.Mechanism;
import org.apache.shale.remoting.XhtmlHelper;
private static XhtmlHelper helper = new XhtmlHelper();
public void encodeEnd(FacesContext context, UIComponent component) throws
    IOException {
   ....
   //shale remoting resource retrieval
   helper.linkJavascript(context, component, writer,
      Mechanism.CLASS_RESOURCE, "/META-INF/fileupload/fileupload.js");
   helper.linkStylesheet(context, component, writer,
      Mechanism.CLASS_RESOURCE, "/META-INF/fileupload/fileupload.css");
}
```

Markup Rendered to Page:

```
<script type="text/javascript" src="/petstore/faces/static/META-
    INF/fileupload/fileupload.js"></script>
<link type="text/css" rel="stylesheet" href="/petstore/faces/static/META-
    INF/fileupload/fileupload.css" />
```

Delegating to Dynamic Resources

```
import org.apache.shale.remoting.Mechanism;
import org.apache.shale.remoting.XhtmlHelper;
private static XhtmlHelper helper=new XhtmlHelper();
public void encodeBegin(FacesContext context, UIComponent component)
    throws IOException {
   ....
   String fileUploadCallback = helper.mapResourceId(context,
    Mechanism.DYNAMIC_RESOURCE,
    "/bpui_fileupload_handler/handleFileUpload");
   outComp.getAttributes().put("onsubmit", "return
    bpui.fileupload.submitForm(this, '" + retMimeType + "', '" + retFunction + "','" +
    progressBarDivId + "', '" + fileUploadCallback + "')");
}
```

Markup Rendered to Page:

```
onsubmit="return bpui.fileupload.submitForm(this, 'text/xml',
    'bpui.fileupload.defaultRetFunction','progress3x',
    '/petstore/faces/dynamic/bpui_fileupload_handler/handleFileUpload')"
```

Extending Base Renderer Functionality

- Adding AJAX functionality
- Need new Tag Handler
- Don’t want to write a renderer from scratch
- Look up default renderer and delegate

Extending Base Renderer Functionality

```
public void encodeBegin(FacesContext context, UIComponent
    component)  {
   ...// lookup using family and rendererType
   Renderer
    baseRenderer=context.getRenderKit().getRenderer("javax.faces.For
    m", "javax.faces.Form");
   baseRenderer.encodeBegin(context, component);
}
public void encodeEnd(FacesContext context, UIComponent
    component)  {
   ... // lookup using family and rendererType
   Renderer
    baseRenderer=context.getRenderKit().getRenderer("javax.faces.For
    m", "javax.faces.Form");
   baseRenderer.encodeEnd(context, component);
}
```

Component Library Summary

- Accessing Resources through Renderers and PhaseListerns can impact performance
- Preferred hybrid approach
- Preferred zero setup time (other than adding jar)
- Still exploring alternative approaches

Agenda

AJAX Overview

Demo

Application Design

JavaScript Guidelines

JSF Approach

Summary

Summary

- AJAX and Java EE 5 are Complimentary
- Use Page as the Application Architecture
- Follow MVC in your application
- Use Proxy Style or REST APIs for Mashups
- Use JavaScript Conventions
- Leverage an Existing AJAX Library
- Wrap AJAX in Java Server Faces for Re-Use
- Try the Java BluePrints AJAX Components!

For More Information

- BluePrints Projects on Java.net
  http://blueprints.dev.java.net/
- Java Pet Store 2.0
  https://blueprints.dev.java.net/petstore
- AJAX Components
  http://blueprints.dev.java.net/ajaxcomponents.html

- Talk: BluePrints for Java EE 5, TS-1969
  Wednesday, 4:00pm–5:00pm
- BOF-2594 Thursday Night, 7:30pm–8:20pm

Q&A

Sean Brydon, Greg Murray,
Inderjeet Singh, Mark Basler

Java™ EE 5 BluePrints for AJAX-Enabled Web 2.0 Applications

Sean Brydon, Greg Murray, Inderjeet Singh, Mark Basler

Java BluePrints
Sun Microsystems, Inc.
http://blueprints.dev.java.net/

TS-1615