Web Application Frameworks and Mashups

Martin Vasko
Distributed Systems Group
Institute of Information Systems

http://www.infosys.tuwien.ac.at/
Outline

- Part 1 – Web Application Frameworks and Mashups
  - Web Application Frameworks
  - Mashups
  - Web 2.0 <> SOA
  - Rich Internet Applications
  - Sample Framework: Rails and Flex

- Part 2 – RESTful Web Services
Web Application Frameworks
Trends

Full stack (JavaEE, Ruby on Rails, Seaside, …)

Web languages (PHP, Active Server Pages)

Common Gateway Interface (CGI)
Architecture

- Model
  Domain specific representation of the information

- View
  Renders the model in a suitable form for interaction

- Controller
  Processes and responds to events
The View and AJAX

AJAX
Asynchronous JavaScript and XML

XMLHttpRequest - Object
AJAX Sample

```html
<form id="inputForm">
  <input type="text" value="" onkeyup="makeRequest()" id="field" />
  <input type="text" value="" id="result" />
</form>

function makeRequest() {
  curPLZ = document.forms[0].elements[0].value
  http_request = false;
  http_request = new XMLHttpRequest();
  if(http_request.overrideMimeType) {
    http_request.overrideMimeType('text/xml');
  }
  http_request.onreadystatechange = showResponse;
  url = "http://localhost:8080/Ajax/Servlet" + "?plz" + curPLZ;
  http_request.open('GET', url, true);
  http_request.send( null );
}
```
public class PLZServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();

        String param = request.getParameter("plz");
        String result = "?";
        // Database access
        result = "Wieden - Vienna, Austria";
        // render response
        out.println("<?xml version="1.0"?><root>" + result + "</root>");
    }
}
function showResponse() {
  if(http_request.readyState == 4) {
    if(http_request.status == 200) {
      var xmlDoc = http_request.responseXML;
      var root_node = xmlDoc.getElementsByTagName('root').item(0);
      document.forms[0].elements[1].value = root_node.firstChild.nodeValue;
    } else {
      alert('Error processing the request');
    }
  }
}
AJAX Design Patterns

- Usability Principles
  - Follow web standards
  - The browser is not a desktop
  - Provide affordances
  - Smooth, continuous interaction
  - Customization

- Software Design Principles
  - Embrace JavaScript
  - Accept workarounds where necessary
  - Develop for compatibility
  - Reduce bandwidth
  - Deal with latency
  - Multiple tiers
AJAX Design Patterns

- Foundation Technology Patterns
  - Web Remoting, Dynamic Behaviour, …

- Programming Patterns
  - Web Services, DOM Population, Code generation and reuse

- Functionality and Usability Patterns
  - Widgets, Page Architecture, Visual Effects, Functionality

- Development Patterns
  - Diagnosis, Testing
AJAX Frameworks

- **Prototype** [http://www.prototypejs.org](http://www.prototypejs.org)
  - A base library for AJAX enabled Web Sites
  - Basis for other frameworks
- **DoJo** [http://dojotoolkit.org](http://dojotoolkit.org)
  - Very powerful framework
  - DoJo – Foundation (IBM, AOL, etc.)
  - very popular
- **Script.aculo.us** [http://script.aculo.us](http://script.aculo.us)
  - JavaScript effect framework with very good AJAX support
  - Add-on for prototype
  - Ruby on Rails Integration
- **Ext** [http://extjs.com](http://extjs.com)
  - Another JavaScript effect framework with AJAX support
  - Powerful DOM processing
- ... and many more
Mashups
Mashup

- Originates from the musical domain (remixes, etc.)
- Web Application composing existing Services and data
- Use of open APIs from other Web Applications
- Sample: Netvibes
Mashup Properties

- Aggregate content from more than one source
  - Public Web Service APIs (flickr.com, maps.google.com, …)
  - Data feeds from other providers (google search, news feeds, …)
  - User-provided information (wikipedia)

- Lightweight programming effort
  - Numerous toolkits (mostly based on JavaScript and HTML)

- Interactive Web Application

- Ad-hoc composition
Mashup Architecture

- **Web Browser**
- **Web Service APIs**
- **Web Data Feeds**
- **Client**
- **Local Database**

Connections:
- AJAX
- HTTP/XML
- HTTP/JSON
- HTTP/SOAP/RSS
- HTTP/RSS/ATOM
Mashup Client - side

Web Browser

Mashup (HTML, JavaScript, Flash, ...)

HTTP/JPEG

HTTP/RSS/ATOM

HTTP/XML/JSON

Web Service APIs

Web Data Feeds

Web Server

Security Policies!
Mashup Server - side

Web Browser

Mashup (HTML, JavaScript, Flash, …)

Security Policies!

Web Server

Mashup (PHP, Ruby, JSP, ASP.NET, …)

HTTP/RSS/ATOM

SOAP

Web Service APIs

Web Data Feeds
Web 2.0 <> SOA
Web 2.0 <-> SOA

- Classification of Web 2.0 applications
  - Community
  - Plattform
    - http://pipes.yahoo.com/pipes
  - Online Collaboration
    - http://docs.google.com

- Classification of SOA applications
  - Cross-organizational
  - Integrating heterogene systems
  - Development principle
Web 2.0 <-> SOA

- **Commonalities**
  - Encapsulate functionality
    - Access functionality through interfaces
    - Hide internal complexity
  - Composeability
    - Web 2.0: Web Sites enhanced by external Data feeds
    - SOA: Different services composed to a new, more complex service
  - Agility
    - Flexible and easily adaptable architecture
    - Enterprise 2.0
  - User empowerment
    - Users can modify applications
Web 2.0 <> SOA

- **Differences**
  - Standardization (or a lack thereof)
    - SOA is well standardized
    - Web 2.0 is not
    - Governance
  - **Social aspects**
    - SOA couples disparate business functions and encourage collaboration indirectly
    - Web 2.0 enforces it, Human Participants are essential parts of the value chain
Web 2.0 ↔ SOA

- Web 2.0 technical fundament
  - XML
    - structure and specify data
  - RSS
    - Easy and efficient aggregation
  - REST (> Part 2)
  - Ajax

- SOA
  - following lectures …
Rich Internet Applications
Rich Internet Applications

- **Pros**
  - Usability
  - Responsive User Interface
  - zero Installation
  - Secure environment
  - less server requests

- **Cons**
  - Longer download
  - may grow resource hungry
  - Runtime requirements (Sandboxes: Java, Flash, …)
Rich Internet Applications

- **Technologies**
  - JavaScript
    - Ext JS
    - Script.aculo.us
  - Java Applets
  - Microsoft Silverlight
    - XAML (eXtensible Application Markup Language)
  - Mozilla XUL
    - XML User Interface Language
  - Flash
    - Flex (ActionScript 3.0 & MXML)
**Flex Sample**

**Source**
- ActionScript 3.0
- MXML
- Assets (PNG, GIF, etc.)

**CodeGen**
- ActionScript + Generated ActionScript
- Assets (PNG, GIF, etc.)

**Flash Player**
- AVM2 Bytecode + Assets
<?xml version="1.0" encoding="utf-8"?>

    <mx:HTTPService id="listBugs" url="http://localhost:3000/bugs/list" useProxy="false" method="GET"/>
    <mx:HTTPService id="updateBug" url="http://localhost:3000/bugs/update" useProxy="false" method="POST" result="updateBug(result)"/>
    <mx:HTTPService id="deleteBug" url="http://localhost:3000/bugs/delete" useProxy="false" method="POST" result="deleteBug(result)"/>
    <mx:HTTPService id="createBug" url="http://localhost:3000/bugs/create" useProxy="false" method="POST" result="createBug(result)" contentType="application/xml">
        <mx:request xmlns=""/>
        <bug>
            <reportedby>{reportedby.text}</reportedby>
            <assignedto>{assignedto.text}</assignedto>
            <description>{description.text}</description>
            <status>{status.text}</status>
            <priority>{priority.text}</priority>
        </bug>
    </mx:request>
</mx:HTTPService>

<mx:Script>
    <![CDATA[
        [Bindable]
        private var statusArray:Array = ["Opened", "Assigned", "Closed"];

        [Bindable]
        private var priorityArray:Array = ["Blocker", "Critical", "Major", "Minor", "Trivial"];

        private function clearForm():void
        {
            reportedby.text = "";
            assignedto.text = "";
            description.text = "";
            status.selectedIndex = 0;
            priority.selectedIndex = 0;
        }
    ]]>]
</mx:Script>
<mx:VDividedBox x="0" y="0" height="100%" width="100%">
    <mx:Panel width="100%" height="376" layout="absolute" title="Create/Update Bugs">
        <mx:Form x="10" y="10" width="930" height="230">
            <mx:FormItem label="Reported by">
                <mx:TextInput width="220" id="reportedby" text="{bugs_dg.selectedItem.reportedby}"/>
            </mx:FormItem>
            <mx:FormItem label="Assigned to">
                <mx:FormItem label="Description">
                    <mx:FormItem label="Status" width="287">
                        <mx:FormItem label="Priority">
                            </mx:FormItem>
                        </mx:FormItem>
                    </mx:FormItem>
                </mx:FormItem>
            </mx:FormItem>
        </mx:Form>
    </mx:Panel>
    <mx:ControlBar horizontalAlign="right">
        <mx:Button label="Clear" click="clearForm()'"/>
        <mx:Button label="Update" click="sendBugUpdate(); clearForm()'"/>
        <mx:Button label="Create" click="createBug.send(); clearForm()'"/>
    </mx:ControlBar>
    <mx:Panel width="100%" height="444" layout="absolute" title="Bugs">
        <mx:DataGrid x="0" y="0" width="100%" height="100%" id="bugs_dg" dataProvider="{listBugs.bugs.bug}'">
            <mx:columns>
                <mx:DataGridColumn headerText="Reported by" dataField="reportedby'"/>
                <mx:DataGridColumn headerText="Assigned to" dataField="assignedto'"/>
                <mx:DataGridColumn headerText="Description" dataField="description'"/>
                <mx:DataGridColumn headerText="Status" dataField="status'"/>
                <mx:DataGridColumn headerText="Priority" dataField="priority'"/>
            </mx:columns>
        </mx:DataGrid>
    </mx:Panel>
</mx:VDividedBox>
Sample Framework: Rails
Build your Rails application …
> rails demo
> cd demo/

... and generate a model …
> ruby script/generate model bug

... and generate a controller …
> ruby script/generate controller bugs

... and an action
```ruby
class BugsController < ApplicationController
  def create
    end
  def list
    end
  def update
    end
  def delete
    end
end
```
Quickstart

... define a view ...
<html>
  <head>
    <title>Hello from Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
  </body>
</html>

Save it under .../demo/app/views/say/hello.rhtml

... and fire up the built in web server ...
demo/> ruby script/server
=> Booting WEBrick...
Quickstart

```ruby
class SayController < ApplicationController
  def hello
    end
  end

<html>
  <head>
    <title>Hello, Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
  </body>
</html>
```
Resources

- Dave Thomas and David Heinemeier Hansson
  „Agile Web Development with Rails“
  *The Pragmatic Programmers*
- Michael Mahemoff
  “Ajax Design Patterns”
  *O’Reilly, Theory in Practice*
- E. M. Maximilien and A. Ranabahu
  “The Programmable Web: Agile, Social, and Grassroots Computing”
  *In Proceedings of 1st IEEE International Conference on Semantic Computing special session on The World of Semantics beyond the Semantic Web, Irvine, CA, 2007*
Links

- The Programmable Web
  http://www.programmableweb.com/
- AJAX
- Mashups [German]
  http://www.frankwestphal.de/Mashups-Remixme.html
RESTful Web Services

Philipp Leitner
Distributed Systems Group
Institute of Information Systems
What is REST?

REST

... is an Acronym for Representational State Transfer

... is an Architectural Style for Distributed Systems
  No Communication Protocol
  No Web Services Standard
  No Deployable Technology

... is the Basic Architecture of the WWW

That Means: REST is the Foundation of the Largest Deployed Information System Today!
What are the Goals of REST?

- Many Design Goals

- But Among the Most Important are
  - Scalability
  - Simplicity of Interfaces
  - Independence of Components

- Other Goals are Subordinated
  - e.g., 'Enterprise' Features (Transactions, …)
What does REST Look Like?

Main Characteristics:

- **Resource-Orientation**
  - Key Elements of Any RESTful Service are Resources
  - Activities are not Explicitly Modelled

- **Statelessness**
  - Every Request is Self-Contained

- **Uniform Interface**
  - Every Resource is Accessed Through the Same Interface

- **Naming**
  - Every Resource is Associated With an Unique and Descriptive Name

- **Layering**
  - Intermediaries Can be Inserted Transparently
  - Proxies, Caches, …
How is REST Implemented?

- **Main Protocols:**
  - **RFC 2068 / 2616**
    - HTTP
    - Communication protocol
    - Today Mostly RFC 2616 (HTTP 1.1)
  - **RFC 2396**
    - URI
    - Resource Identifiers
How is REST Implemented?

- **Characteristics Revisited:**
  - **Resource-Orientation**
    - Resources are Represented in a MIME Type
    - e.g., `text/html`, `text/plain`, `image/jpeg`, …
  - **Statelessness**
    - HTTP is a Stateless Protocol
    - (if you Forget About Cookies …)
  - **Uniform Interface**
    - HTTP Interface
    - GET / POST / PUT / HEAD / DELETE
  - **Naming**
    - Every Resource is Identified by a URI
    - URIs are Usually Relatively Descriptive
    - e.g., `http://en.wikipedia.com/wiki/REST`
  - **Layering**
    - HTTP Works Over Caches, Proxies, Gateways, Routers…
From REST to RESTful Web Services

“The problem is, most of today‘s web services have nothing to do with the Web. (…) Today‘s web service architectures reinvent or ignore every feature that makes the Web successful.“

[RESTFul Web Services]

‘Simple‘ Solution:
- Create Web Services That are In Line With the Web
- Create Web Services According to the REST Principles
RESTful Web Service Properties

- **Resources** are Important
  - Main Focus is on CRUD
- **Addressing** is Important
  - Every Logical or Physical Resource Needs a URI
- **Statelessness** is Important
  - No Transactions Along >1 Service Invocations

- REST is Just a Way of Building Services
  - No Concrete Protocol
  - RESTful Services Employ 'ad hoc' Protocols
RESTful Web Service Technology

- HTTP + URI
  - Every RESTful Service Uses HTTP
  - Plain TCP / SMTP / … are Not Supported
  - (Hence the Name ‘Web‘ Services)

- XML
  - Prevalently Used for Resource Representation
  - Other Possibilities:
    - JSON
    - HTML
    - Plain Text
    - …
RESTful Web Service Technology

- JSON (JavaScript Object Notation)
  - Alternative Resource Representation Format
  - Light-Weight Alternative to XML
  - Integrates Particularly Well with JavaScript (eval() )

```json
{"REST": {
 "repr": [
{"name": "xml",
 "standard": "rfc3076"}
{"name": "json",
 "standard": "rfc4627"}
 ]
}
```

- Serialized JavaScript Values
RESTful Web Service Technology

- JSON Security
  - Don‘t EVER Use eval() on JSON Data From Sources You Don‘t Control !!!
    - XSS (Cross-Side Scripting) !
    - SQL Injection !
RESTful Web Service Technology

- **WADL**
  - Web Application Description Language
  - **Machine-Processable** REST Service Description
    - REST Equivalent to WSDL
  - Allows to Define
    - What Resources Exist in a Service?
    - How are Resources Represented?
    - What HTTP Methods Can be Used?
    - What Parameters are Supported?
  - Foundation of 'Formal' Usage of REST Services
  - Currently **Very Rarely Used**
RESTful Web Service Technology

```xml
<application xmlns="http://research.sun.com/wadl"
xmlns:ex="http://my.example.namespace/">
  <grammars>
    <include href="MyServiceSchema.xsd" />
  </grammars>
  <resources base="http://my.search.com/MySearch/Resource1/">
    <resource uri="mySearch">
      <method href="#search" />
    </resource>
  </resources>
  <method name="GET" id="search">
    <request>
      <param name="searchstring" style="query" 
        type="xsd:string" required="true" />
    </request>
    <response>
      <representation mediaType="application/xml"
        element="ex:MySearchResult" />
    </response>
  </method>
</application>
```

Defines a Service That Accepts a GET Request Like
http://my.search.com/MySearch/Resource1?searchstring=WADL
And Returns a XML File According to MyServiceSchema.xsd
Example 1: Amazon

http://ecs.amazonaws.com/onca/xml?Service=AWSECommerceService&AWSAccessKeyId=[KEY]&Operation=ItemSearch&SearchIndex=PCHardware&Title=ATI&Version=2007-07-16

- <Item>
  - <ASIN>B000SMZ14U</ASIN>
  - <DetailPageURL>
    http://www.amazon.com/gp/product/B000SMZ14U/ref=s9_spinf_m2%26as_sc_corEH推荐ID-2005%26oscmID-1559
  </DetailPageURL>
  - <ItemAttributes>
    - <Manufacturer>Diamond</Manufacturer>
    - <ProductGroup>CE</ProductGroup>
    - <Title>
      Diamond Viper ATI Radeon HD 2600 XT PCIe 512MB GDDR3 Video Card
    </Title>
  </ItemAttributes>
</Item>

The Trick is - There is No Trick.
Example 2: Flickr

http://api.flickr.com/services/rest/?method=flickr.interestingness.getList&api_key=[key]&per_page=5

```xml
<rsp stat="ok">
  <photos page="1" pages="100" perpage="5" total="500">
    <photo id="2061250569" owner="84745314@N00" secret="b0986fbcdf" server="2228"
    <photo id="2061958693" owner="82542781@N00" secret="f105fcb614" server="2007"
    <photo id="2063995632" owner="62071753@N00" secret="3fa0c16084" server="234"
    <photo id="2061535333" owner="24183489@N00" secret="2e4b0a27da" server="216"
    <photo id="2063099936" owner="73744298@N00" secret="d49c84c754" server="207"
  </photos>
</rsp>
```
How ‘RESTful’ are REST Services Today?

- Often Seen Misconceptions:
  - **One Endpoint Catches All**
    - http://my.example.com/rest/endpoint
  - **RPC Semantics Encoded in HTTP**
    - http://my.example.com/invoke/methodA?param1=1&param2=2
  - **HTTP GET is Used Even for ‘Non-Safe’ Operations**
    - http://my.example.com/photo12345/delete
  - ‘Methods‘ are Encoded as GET or POST Parameters
    - http://my.example.com/photo12345/?method=showMetaData
  - **Authentication via API Keys**
    - http://my.example.com/photo/12345/?key=philippleitner
    - Security? Non-Reputiation?
  - **URIs are Often not Human-Readable**
    - http://my.example.com/photo/aaaabbb1234ksdhf76
Hybrid Services

“(…) web services that fit somewhere in between the RESTful web services and the purely RPC-style services. These services are often created by programmers who know a lot about real-world web applications, but not about the theory of REST.”

[RESTful Web Services]

- Most **Real-Life RESTful Services**
  - Have Some Properties of REST
  - But Also Some RPC-Like Features
  - Trade-Off: REST Theory and Real-World Requirements

- Common Terms:
  - POX over HTTP, HTTP+POX
    - (Plain Old XML over HTTP)
  - Hybrid Services
Advantages

- Lightweight
- Easy to Use
- Easy to Test

- Proven Scalable
- Reusing Existing Web Infrastructure
  - Caches
  - HTTPS
  - ...

- 'Web Services for the Real World'
Disadvantages

- How Would You Do …
  - Authentication (Besides Realms)?
  - Security (Besides HTTPS)?
  - Transactions?
  - Reliable Messaging?
  - Policies?
  - Service Orchestration?

- ’Enterprisy‘ Features are Not Considered
Disadvantages

- 'REST an Architectural Style, Not a Communication Protocol'  
  - So - What's the Communication Protocol?

- 'REST Exposes an Uniform Interface'  
  - And how do I Learn the Semantics of This Interface for a Specific Service?

- 'REST is Simple to Implement'  
  - And What Happened to the Framework Support?
'Format Wars' are No Stranger in Computer Science

- 'It's just Emacs vs. VI all over again…'
- Often Purely Ideological Debates
- From a Technical Point the Similarities are Predominant
Activities vs. Resources

**SOAP**
Client → Activity 1
Client → Activity 2
Client → Activity 3

**REST**
GET
PUT
POST
DELETE

**SOAP Services Model**
ACTIVITIES
e.g., a Bank Transaction

**REST Services Model**
DATA
e.g., a Lyrics Database

Enterprise vs. Web 2.0

- **SOAP**
  - Main Focus:
    - SOA
    - EAI
  - Supports Different Protocol Bindings
  - WS-* Stack Supports Enterprise Features
  - Enterprise-Level Service Frameworks Available
    - Apache CXF
    - Apache Axis (2)
    - Microsoft WCF

- **REST**
  - Main Focus:
    - Web 2.0
  - Simple Usage
  - Light-Weight
  - 'Per API' Tool Support
Generality vs. Simplicity

- **SOAP**
  - Idea:
    - Have a Spec. for Every Possible Use Case
    - Cover Everything (and Introduce Extensibility to Cover Everything Else)
  - Web Services as the Silver Bullet of Distributed Computing

- **REST**
  - Idea:
    - Keep the 'Standards' as Simple and High-Level as Possible
    - Let Engineers Figure Out the Details on Demand
  - Web Services as Simple Web Integration Technology
Further Reading

- REST Wiki
  - [http://rest.blueoxen.net/cgi-bin/wiki.pl](http://rest.blueoxen.net/cgi-bin/wiki.pl)

- RESTful Web Services
  (O’Reilly, ISBN 0596529260)

- Fielding (PhD Thesis)

- Resource-Oriented vs. Activity-Oriented