Lightweight REST Framework for Java
Outline

- REST Architectural Style
- Restlet Project
- Restlet Programming
- Restlet & Other Technologies
- Deployment Options
- Overstock.com Experience
- Q & A
REST Architectural Style
What is REST?

- **RE**presentational **State** **Transfer**
- Formalized by Roy Fielding in his PhD Dissertation
- Primarily applicable to distributed hypermedia systems
- Think of it as resource-orientation
  - Resources represent the domain concepts
Roy’s Motivation for REST

- Architectural model for how the Web was designed and *should* work
- Serves as a guide for Web standards
- REST has been applied to:
  - Describe the desired Web architecture
  - Help identify existing problems
  - Compare alternative solutions
Your Motivation for REST

- Take advantage of what the Web does well
  - Simplicity
  - Scalability
  - Performance
  - Ease of use
- So much nicer than the alternatives
  - SOAP & WS - *
- Unifies Web Sites and Web Services into consistent Web Applications
A Style, Not a Standard

- But REST guides the use of standards
- For example:
  - HTTP (Connector)
  - URI (Resource)
  - XML, HTML, GIF, etc. (Representations)
    - text/xml, text/html, image/gif, etc. (Media types)
- The Web is a REST system
What is an Architectural Style?

“...a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style.”
- Dr. Roy Fielding

- Some Network-based Architectural Styles
  - Pipe-and-Filter
  - Client-Server
  - Layered
  - Virtual Machine
  - Code on Demand
  - Mobile Agent
  - Event-based
  - Distributed Objects
REST Architectural Style

- Composition of styles that gains their benefits:
  - **Client-Server** - separation of concerns, scalability
  - **Layered** – allows intermediaries (proxies, firewalls) without affecting interfaces
  - **Stateless** – scalability
  - **Cacheable** – reduces payload & latency
  - **Pipe-and-Filter** – dynamic component connection
Representational State Transfer

- Imagine an application as a network of web pages
  - Virtual state-machine
- The user progresses by selecting links…
  - State transitions
- …resulting in the next page…
  - Representing the next state of the application
- …being transferred to the user
State Transitions in REST

- Numbers are resources (URIs, eg., hyperlinks)
- Letters are representations (HTML, XML, jpg, etc), that may contain hyperlinks to next states

![State Transition Diagram]

- $S_0$ to $S_1$ with label 1/a
- $S_0$ to $S_2$ with label 2/b
- $S_1$ to $S_2$ with label 3/c
- $S_2$ to $S_3$ with label 4/d
- $S_2$ to $S_4$ with label 3/c
- $S_3$ to $S_4$ with label 3/c
- $S_4$ to $S_5$ (out of frame)

**State**

1. **S0**
2. **S1**
3. **S2**
4. **S3**
5. **S4**
6. **S5**

**Transition Labels**

- 1/a
- 2/b
- 3/c
- 4/d

[State Transition Diagram]
Resources

- A Resource should be a fixed target of a URI
- Is semantic: "Today's weather in Park City"
- The URI-to-Resource mapping shouldn't change, but the representation can
- Resources may map to multiple representations, called variants
  - Example: png, gif, jpg are variant representations of an image
  - Content negotiation selects the best variant
Uniform Interface

- Supports the constraints of Client/Server, and Layered architectural styles
- Resources are manipulated by HTTP methods
  - GET – retrieve a resource
  - PUT – create a resource
  - POST – update (create if necessary) a resource
  - DELETE – delete a resource
Interoperability on a Global Scale

- REST advocates (and constrains the use of) existing Web standards:
  - **URI** – how resources are named and referenced
  - **Methods** – how resources are manipulated
  - **HTML, XML, GIF, etc** – how resources are represented
  - **Media types (text/plain, etc)** – metadata for representations
Cachability

- Reduces latency, increases scalability through reduced bandwidth utilization
- REST architectural constraints allow caches to be injected anywhere in the application
- A cache can return copy in response to a GET, therefore prefer GET over POST
Principles of REST

- Some principles are still debated
- Do what makes sense for your application, but be conscious of the tradeoffs

“You're pirates. Hang the code, and hang the rules. They're more like guidelines anyway.”
– Elizabeth (Pirates of the Caribbean)
Principles of REST

- URIs refer to resources, not representations
  - www.overstock.com/home+and+garden
  - www.overstock.com/home+garden.html
- Resources are nouns, not verbs
- GET never has side effects, and anything that has no side effects should use GET
- Use links in responses enable state transfer
Principles of REST

- URI “/” means parent-child or whole-part relationship
- Avoid query strings in URIs (debatable)
  - www.overstock.com/products/id=123
  - www.overstock.com/products/123
- In the later case, the relationship is clear and can be extended for subresources
- Provide data to clients via gradual unfolding
References

- Roger L. Costello, [www.xfront.com](http://www.xfront.com)
- Paul Prescod, [www.prescod.net/rest](http://www.prescod.net/rest)
Restlet Project
What is Restlet?

- An open source REST framework for Java
- A good mapping of REST principles
- Founded by Jérôme Louvel, Noelios Consulting, Paris, France
  www.restlet.org
- Built in response to:
  - Need for a simple, RESTful web application framework
  - Servlet limitations
Restlet Programming

Lightweight REST framework for Java
Restlet Framework

**Restlet API** – Supports REST call handling

**Extensions** – For integrating external technologies (JDBC, JSON, alternate containers, connectors, template engines, etc.)

**SPI** – Plugin point for alternate implementations

**Restlet Implementation** – Currently just Noelios Engine
A REST Architecture

Lightweight REST framework for Java

Overstock.com
slandis@overstock.com

03/15/2007

Slide 24

A REST Architecture
File Browsing Example

- Component contains VirtualHosts, Applications and Server and Client connectors
- Default Host is ‘built-in’
- Notice the beginnings of a pipes-and-filters architecture within a client-server architecture

```
Component
Restlet Component
HTTP Server localhost:8182
Application
Default Host
Directory
File Client
```

Lightweight REST framework for Java

Overstock.com
slandis@overstock.com

03/15/2007

Slide 25
Application Class

- Contains your “application” logic
- Contains useful services that can be overridden, such as:
  - connectorService
  - decoderService
  - statusService
Directory Class

- Finder of file system resources
- Automatic content negotiation similar to Apache HTTP server
  - Selects best representation based on
    - available variants
    - client capabilities and preferences
public class FileServer implements Constants {

    public static void main(String[] args) throws Exception {
        Component component = new Component();
        component.getServers().add(Protocol.HTTP, 8182);
        component.getClients().add(Protocol.FILE);

        Application application = new Application(component.getContext()) {
            @Override public Restlet createRoot() {
                Directory directory = new Directory(getContext(),
                        "file://" + ROOT);
                directory.setListingAllowed(true);
                directory.setDeeplyAccessible(true);
                return directory;
            }
        };

        component.getDefaultHost().attach("", application);
        component.start();
    }
}
VirtualHost Class

- Router of calls from Server connectors to Restlets; typically an Application
- Defined along three properties:
  - Request's "hostRef"
  - Request's "resourceRef"
  - Response's "serverInfo"
- Host multiple applications in a single JVM
  - Same IP address shared by several domain names
  - Same domain name load-balanced across several IP addresses
Virtual Hosts Example

- A VirtualHost routes requests to Applications by regular expression matching

- Grey items are included for illustration
VirtualHost Example

```java
public class VirtualHostServer implements Constants {

    public static void main(String[] args) throws Exception {
        Component component = new Component();
        component.getServers().add(Protocol.HTTP, 8182);
        component.getClients().add(Protocol.FILE);

        VirtualHost vh1 = new VirtualHost(component.getContext());
        // Host names must be distinguished and not made up.
        vh1.setHostDomain("localhost");

        Application application1 =
            new Application(component.getContext()) {
                @Override public Restlet createRoot() {
                    Directory directory = new Directory(getContext(),
                        DOC_URI);
                    return directory;
                }
            };
    }
```
VirtualHost vh2 = new VirtualHost(component.getContext());
vh2.setHostDomain("127.0.0.1");
Application application2 =
    new Application(component.getContext()) {
        @Override public Restlet createRoot() {
            Restlet jarRestlet = new Restlet(getContext()) {
                @Override public void handle(Request request,
                                            Response response) {
                    File file = new File(JAR_PATH);
                    FileRepresentation frep =
                        new FileRepresentation(file,
                                                 MediaType.APPLICATION_JAVA_ARCHIVE, 1000);
                    response.setEntity(frep);
                    response.setStatus(Status.SUCCESS_OK);
                }
            };
            return jarRestlet;
        }
    };

...Continued
VirtualHost vh3 = new VirtualHost(component.getContext());
vh3.setHostDomain("C1LPT083");
Application application3 =
    new Application(component.getContext()) {
        @Override public Restlet createRoot() {
            Restlet pageRestlet = new Restlet(getContext()) {
                @Override public void handle(Request request, Response response) {
                    File file = new File(PAGE_PATH);
                    FileRepresentation rep =
                        new FileRepresentation(file, MediaType.TEXT_HTML, 1000);
                    response.setEntity(rep);
                    response.setStatus(Status.SUCCESS_OK);
                }
            };
            return pageRestlet;
        }
    };

...Continued
…Continued

vh1.attach("/docs/", application1);
vh1.attach("/docs/deprecated", application3);
    component.getHosts().add(vh1);
vh2.attach("/jar", application2);
    component.getHosts().add(vh2);
vh3.attach("/page", application3);
    component.getHosts().add(vh3);
    component.start();
}
Restlet Class

- Uniform interface class
  - Get, Put, Post, Delete
- Context
- Life cycle support
- Its subclasses implement specific ways to process calls
Restlet Class

Uniform

Restlet

Connector  Application  Router  Finder  Component  Filter

Redirector

Client  Server  VirtualHost  Directory  Guard  Route
Router Class

- **Restlet** for routing calls to one of the attached routes (e.g., to another Restlet)
- **attach(pattern, Restlet)**
  - Creates a route based on URI patterns matching the beginning of a the resource reference's remaining part
Example Application

- Same host/port
- Routes based on URI
Router Example

```java
public class RouterServer implements Constants {
    public static void main(String[] args) throws Exception {
        // Initialize connectors as before...

        Application application =
                new Application(component.getContext());
        @Override public Restlet createRoot() {
            Router router = new Router(getContext());
            // Create the Restlets as before...
            router.attach("/docs", directory);
            router.attach("/jar", jarRestlet);
            router.attach("/page", pageRestlet);
            return router;
        }
    }

    component.getDefaultHost().attach("", application);
    component.start();
}
```
Router URI Patterns

- URI Template Spec for variables
- Example URI patterns:
  - /docs/ to display static files
  - /users/{user} to display a user’s account
  - /users/{user}/orders to display the orders of a particular user
  - /users/{user}/orders/{order} to display a specific order
URI Routing

http://host:8080/server/users/123/orders/456

- **Router** sees: /users/123/orders/456
- **Router** sees: /orders/456
Advanced Router

- A Route can compute a score for each call depending on various criteria

- Several routing modes are supported:
  - Best match (default)
  - Round robin
  - Random match
  - First match
  - Last match
  - Custom
Round Robin Example

```java
@Override public Restlet createRoot() {
    Router router = new Router(getContext());
    Restlet restlet1 = new Restlet(getContext()) {
        @Override public void handle(Request request, Response response) {
            StringRepresentation rep =
                new StringRepresentation("Restlet 1");
            response.setEntity(rep);
            response.setStatus(Status.SUCCESS_OK);
        }
    };
    Restlet restlet2 = new Restlet(getContext()) {
        router.setRoutingMode(Router.NEXT);
        router.attach("", restlet1);
        router.attach("", restlet2);
        return router;
    }
```
Filter Class

- Impose before/after handling in call flow

NOTE: A Web Application & Web Service UNIFIED!
Resource

• Remember this?

Resources

• A Resource should be a fixed target of a URI
• Is semantic: "Today's weather in Park City"
• The URI to Resource mapping shouldn't change, but the representation can
• Resources may map to multiple representations, called variants
  – Example: png, gif, jpg are variant representations of an image
  – Content negotiation selects the best variant
Resource Class

- Typically created by a **Finder**
- Selects a **variant** **Representation**
- A final handler of calls in the pipeline
- Not shared between calls; can be thread-unsafe
- Where the RESTful view of your Web application can be integrated with domain objects
  - Databases, beans, other services, etc.
- By default, only the GET method is enabled
Using Resource

- Override REST methods you support: `post()`, `put()`, `delete()`
- Override the matching `allow*()` methods
- Optionally override `handle*()` method for custom content negotiation
- Restlet calls are dynamically dispatched to the `handle*()` methods via introspection
- To support a custom MOVE method
  - `add handleMove()`
@Override public Restlet createRoot() {
    Router router = new Router(getContext());
    router.attach("/users/{username}",
                  UserResource.class);
    router.attach("/users/{username}/bookmarks",
                  BookmarksResource.class);
    Route uriRoute =
        router.attach("/users/{username}/bookmarks/{URI}",
                      BookmarkResource.class);
    uriRoute.getTemplate().getVariables().put("URI",
                                            new Variable(Variable.TYPE_URI_ALL));

    return router;
}
public class UserResource extends Resource {
   ...
   public UserResource(Context context, Request request, Response response) {
      super(context, request, response);
      this.userName = (String) request.getAttributes().get("username");
   ...
   this.user = findUser();
   if (user != null) {
      getVariants().add(new Variant(MediaType.TEXT_PLAIN));
   }
   }
   ...
   @Override public boolean allowDelete() { return true; }
   @Override public boolean allowPut() { return true; }
   ...
@Override
public Representation getRepresentation(Variant variant) {
    Representation result = null;
    if (variant.getMediaType().equals(MediaType.TEXT_PLAIN)) {
        StringBuilder sb = new StringBuilder();
        sb.append("------------
User details
------------

Name: 
    .append(this.user.getFullName()).append('\n');
Email: 
    .append(this.user.getEmail()).append('\n');
result = new StringRepresentation(sb);
    }
    return result;
}

@Override public void put(Representation entity) {
    // Creates a user in a database ...
}
Deployment Options
Many Ways to Deploy

- Deploy as a jar file
- Any Servlet compliant container
  - Tomcat, Jetty
- Native service using Java Service Wrapper
Restlet & Other Technologies
Plays Well With Others

- Various Connectors
  - HTTPS, AJP, Apache HTTP Client, SMTP[S], JDBC, FILE
- Lots of Representations
  - DOM, SAX, XPath, XSLT
  - Template Engine: Velocity, FreeMarker
  - NIO, Apache Upload
- Easy 3\textsuperscript{rd} party integration
  - Eg., Struts, Spring, Hibernate, Acegi, Seam, etc
Overstock.com Experience
Restlet @ Overstock.com

- Created in-house Web Services framework
  - XSD for requests and responses
  - JAXB Filter converts between XML and our object model
- About 6 active developers, more to come
- 3 projects in production, 3 in DEV or QA
- Easy to learn, quick to code, reliable & fast

OVERSTOCK IS HIRING!! TALK TO ME
Not Covered
Lot’s of Other Things

- Finders
- Restlet on the client
- Redirection
- Guards
- NIO
- Logging and error handling
- JSR 311 – REST Annotations
Resource Content Negotiation
GET vs. POST

- The result of a GET is to return a representation of the resource.
- The result of a POST is to post something to a processing resource, which may create a new subordinate resource.
  - In general, the response entity of the POST will describe the status of the method execution, if it succeeded or if it failed.
Returning a Representation from POST

- Example: a complex search request
- Reasons for bending the REST style:
  - URI length overflow – search requests can be very large
  - Information hiding – keep information off the URI
  - Tradition/legacy/migration – “this is how we've done it before”
Returning a Representation from POST

- When POST modifies the target resource, and you want to return the best representation, do this at the end of the `post()` method:

```java
getResponse().setEntity(
    getPreferredRepresentation());
```
Returning a Representation from POST

- To directly return the representation of the created resource, instantiate a new Resource, and manually call:

```java
Resource res = new MyDelegateResource(...);
Representation rep = res.getPreferredRepresentation();
getResponse().setEntityEntity(rep);
```
Finally, if you don't need to take advantage of content negotiation, you can directly set the response entity manually in your `post()` method:

```java
getResponse()
   .setEntity(myJaxbRepresentation);
```
Multi-step Strategies

- **POST /queries**
  - Pass the query document in the request entity
  - Return a status document and a redirectRef with the created query URI

- **GET /queries/264794**
  - Returns the result of the query
  - Idempotent, can be cached

- **DELETE /queries/264794**
  - Cache auto-deletes old queries
Content Negotiation
Types of Content Negotiation

- **Server-driven** - Server picks representation from prior knowledge of client, or uses HTTP header information

- **Client-driven** - Client requests; server returns list of representations; client picks one...requires 2 calls

- **Proxy-driven** - Proxy chooses from a list returned by server, using client preferences

- **URI-specified** – Uses the query string